

# SLAM Handbook

From Localization and Mapping to Spatial Intelligence

## Edited by

Luca Carlone, Ayoung Kim, Timothy Barfoot,  
Daniel Cremers, and Frank Dellaert

**IMPORTANT NOTE ON RELEASE:**

© Cambridge University Press. No reproduction of any part may take place without the written permission of Cambridge University Press.

## Authored by

Henrik Andreasson	Arash Asgharivaskasi	Nikolay Atanasov
Timothy Barfoot	Jens Behley	Jose Luis Blanco-Claraco
Martin Büchner	Cesar Cadena	Marco Camurri
Luca Carlone	Yun Chang	Boris Chidlovskii
Margarita Chli	Henrik Christensen	Javier Civera
Daniel Cremers	Andrew J. Davison	Frank Dellaert
Jia Deng	Gamini Dissanayake	Kevin Doherty
Jakob Engel	Maurice Fallon	Guillermo Gallego
Cédric Le Gentil	Christoffer Heckman	Javier Hidalgo-Carrió
Connor Holmes	Guoquan Huang	Shoudong Huang
Nathan Hughes	Krishna Murthy Jatavallabhula	Michael Kaess
Kasra Khosoussi	Ayoung Kim	Giseop Kim
John Leonard	Stefan Leutenegger	Dominic Maggio
Martin Magnusson	Joshua Mangelson	Hidenobu Matsuki
Matías Mattamala	José M Martínez Montiel	Sacha Morin
Mustafa Mukadam	Jose Neira	Paul Newman
Helen Oleynikova	Lionel Ott	Liam Paull
Marc Pollefeys	Victor Reijgwart	Jerome Revaud
David Rosen	Davide Scaramuzza	Lukas Schmid
Jingnan Shi	Cyrill Stachniss	Niko Sunderhauf
Juan D. Tardós	Zachary Teed	Abhinav Valada
Teresa Vidal-Calleja	Chen Wang	Felix Wimbauer
Heng Yang	Fu Zhang	Ji Zhang
Shibo Zhao		



## Foreword

Simultaneous Localization and Mapping —better known as SLAM— refers to the fundamental problem of building spatial models of an environment while simultaneously determining the position of a robot within that environment. The term itself was first coined in 1995 by Hugh Durrant-Whyte and John Leonard, marking the formalization of a problem that sits at the intersection of robotics, geometry, controls, and probabilistic inference.

SLAM is as elegant as it is formidable. At its core, it addresses the challenge of reasoning over high-dimensional, uncertain, and dynamic systems. The process demands precise spatial inference and robust probabilistic modeling to build coherent maps of the world —maps that must be constructed in real time, often under conditions of noise and ambiguity.

What makes SLAM particularly compelling is its universality. In computer vision, it is mirrored in the problem of Structure from Motion; in robotics, it underpins everything from indoor autonomous navigation to planetary exploration and self-driving cars. Since its inception, SLAM has inspired tens of thousands of research papers, drawing deeply from disciplines as diverse as physics, statistics, computer vision, geometry, controls, and machine learning. Its evolution has catalyzed the development of increasingly capable autonomous systems, able to operate at scale in complex, open-world environments.

This volume brings together contributions from some of the field’s foremost experts and rising stars. The chapters represent the state of the art in SLAM today, reflecting both the depth of theoretical innovations and the breadth of practical applications. From its early formulations based on Kalman filters and Bayesian estimation, SLAM has matured into a rich tapestry of mathematical frameworks —encompassing graph-based optimization, factor graphs, nonlinear least squares, and deep learning-based techniques. Beyond introducing the mathematical foundations of SLAM, this volume provides valuable guidance to the practitioner by discussing real-world use cases ranging from vision-based and LiDAR-based SLAM systems to legged locomotion. It also covers recent developments in Spatial AI, showing how advances in deep learning, differentiable rendering, and large vision

and language models point the way toward representations that provide robots with a rich spatial and semantic understanding of their environment.

The real-world impact of SLAM is unmistakable. Whether embedded in robotic vacuum cleaners or powering fleets of autonomous vehicles, SLAM is now a cornerstone technology that enables intelligent systems to perceive and act in the physical world. It represents one of the most sophisticated and enduring efforts in robotics to date, and continues to shape how machines understand space, uncertainty, and motion.

The SLAM research community is one of the most vibrant and technically rigorous within robotics. Many of today’s leading roboticists began their careers solving SLAM, and have since extended its principles into domains such as state estimation, control, sensor fusion. SLAM stands as one of the defining problems of probabilistic robotics —a field that, through SLAM, has helped usher statistical reasoning into the heart of robotic perception and autonomy.

As contributors to and beneficiaries of this remarkable community, we are proud to introduce this handbook. It offers a comprehensive view of SLAM at its current frontier —building upon three decades of foundational work while pointing the way to the challenges and opportunities ahead. The authors represented here are the next generation of leaders, and their work sets the tone for what is possible when rigorous science meets real-world complexity.

We celebrate their contributions, and we believe this book will serve as both a vital reference and an inspiration for researchers, engineers, and students in the years to come.

Wolfram Burgard, Dieter Fox, and Sebastian Thrun

# Contents

<i>Preface</i>	<i>page</i> xvi
<i>Notation</i>	1
<b>PART ONE FOUNDATIONS OF SLAM</b>	<b>3</b>
<b>I Prelude</b>	<b>5</b>
I.1 What is SLAM?	5
I.2 Anatomy of a Modern SLAM System	7
I.3 The Role of SLAM in the Autonomy Architecture	11
I.3.1 Do We Really Need SLAM for Robotics?	12
I.4 Past, Present, and Future of simultaneous localization and mapping (SLAM), and Scope of this Handbook	15
I.4.1 Short History and Scope of this Handbook	15
I.4.2 From SLAM to Spatial AI	18
I.5 Handbook Structure	19
<b>1 Factor Graphs for SLAM</b>	<b>21</b>
1.1 Visualizing SLAM With Factor Graphs	22
1.1.1 A Toy Example	22
1.1.2 A Factor-Graph View	23
1.1.3 Factor Graphs as a Language	25
1.2 From MAP Inference to Least Squares	26
1.2.1 Factor Graphs for MAPMaximum A posteriori (MAP) Inference	27
1.2.2 Specifying Probability Densities	28
1.2.3 Nonlinear Least Squares	30
1.3 Solving Linear Least Squares	30
1.3.1 Linearization	31
1.3.2 SLAMSimultaneous Localization and Mapping (SLAM) as Least Squares	32
1.3.3 Matrix Factorization for Least Squares	32

1.4	Nonlinear Optimization	34
1.4.1	Steepest Descent	34
1.4.2	Gauss-Newton	35
1.4.3	Levenberg-Marquardt	35
1.4.4	Dogleg Minimization	36
1.5	Factor Graphs and Sparsity	37
1.5.1	The Sparse Jacobian and its Factor Graph	37
1.5.2	The Sparse Information Matrix and its Graph	38
1.5.3	Sparse Factorization	39
1.6	Elimination	40
1.6.1	Variable Elimination Algorithm	40
1.6.2	Linear-Gaussian Elimination	42
1.6.3	Sparse Cholesky Factor as a Bayes Net	45
1.7	Incremental SLAM	47
1.7.1	The Bayes Tree	48
1.7.2	Updating the Bayes Tree	49
1.7.3	Incremental Smoothing and Mapping	51
1.8	Further Readings & Recent Trends	53
<b>2</b>	<b>Advanced State Variable Representations</b>	54
2.1	Optimization on Manifolds	54
2.1.1	Rotations and Poses	55
2.1.2	Matrix Lie Groups	56
2.1.3	Lie Group Optimization	58
2.1.4	Uncertainty and Lie Groups	60
2.1.5	Lie Group Extras	61
2.2	Continuous-Time Trajectories	63
2.2.1	Splines	64
2.2.2	From Parametric to Nonparametric	66
2.2.3	Gaussian Processes	67
2.2.4	Spline and GPs on Lie Groups	70
2.3	Further Readings & Recent Trends	75
<b>3</b>	<b>Robustness to Incorrect Data Association and Outliers</b>	76
3.1	What Causes Outliers and Why Are They a Problem?	76
3.1.1	Data Association and Outliers	76
3.1.2	Least-Squares in the Presence of Outliers	78
3.2	Detecting and Rejecting Outliers in the SLAM Front-end	79
3.2.1	RANDOM SAmple Consensus (RANSAC)	79
3.2.2	Graph-theoretic Outlier Rejection and Pairwise Consistency Maximization	82
3.3	Increasing Robustness to Outliers in the SLAM Back-end	86

	<i>Contents</i>	vii
3.3.1	Iteratively Reweighted Least Squares	90
3.3.2	Black-Rangarajan Duality	91
3.3.3	Alternating Minimization	93
3.3.4	Graduated Non-Convexity	94
3.4	Further Readings & Recent Trends	99
<b>4</b>	<b>Differentiable Optimization</b>	<b>102</b>
4.1	Recap on Nonlinear Least Squares	103
4.2	Differentiation Through Nonlinear Least Squares	104
4.2.1	Unrolled Differentiation	105
4.2.2	Truncated Unrolled Differentiation	107
4.2.3	Implicit Differentiation	108
4.3	Differentiation on Manifold	110
4.3.1	Lie Group Derivatives	110
4.3.2	Differentiation Operations on Manifold	112
4.4	Numerical Challenges of Automatic Differentiation and Modern Libraries	114
4.4.1	Example of Implementation of Differentiable Opti- mization	116
4.4.2	Related Open-source Libraries	117
4.5	Further Readings & Recent Trends	119
<b>5</b>	<b>Dense Map Representations</b>	<b>120</b>
5.1	Range Sensing Preliminaries	120
5.1.1	Sensor Measurement Model	121
5.1.2	Conversion to Point Cloud	122
5.2	Foundations of Dense Mapping	123
5.2.1	Occupancy Maps	124
5.2.2	Implicit Surface and Distance Fields	125
5.2.3	Occupancy Maps or Distance Fields?	127
5.3	Map Representations	127
5.3.1	Explicitness of Target Spatial Structures	127
5.3.2	Types of Spatial Abstractions	128
5.3.3	Data Structures and Storage	133
5.4	Constructing Maps: Methods and Practices	136
5.4.1	Points	136
5.4.2	Surfels	137
5.4.3	Meshes	138
5.4.4	Voxels	139
5.4.5	Gaussian Processes	142
5.4.6	Hilbert Maps	144
5.4.7	Deep Learning in Mapping	145
5.5	Usage Considerations	146

5.5.1	Environmental Aspects	146
5.5.2	Downstream Task Types	147
5.5.3	Summary of Mapping Methods	148
<b>6</b>	<b>Certifiably Optimal Solvers and Theoretical Properties of SLAM</b>	<b>150</b>
6.1	Certifiably Optimal Solvers for SLAM	151
6.1.1	Shor's Relaxation	153
6.1.2	SE-Sync: Certifiably Correct Pose-Graph Optimization	155
6.1.3	Landmark-based SLAM	166
6.1.4	Extensions: Range Measurements, Anisotropic Noise, and Outliers	169
6.2	How Accurate is the Optimal Solution of a SLAM Problem?	174
6.2.1	Cramér-Rao Lower Bound and the Fisher Information Matrix	175
6.2.2	Fisher Information Matrix and Graph Laplacian	177
6.3	Further Readings & Recent Trends	180
<b>PART TWO SLAM IN PRACTICE</b>		<b>183</b>
<b>II</b>	<b>Prelude</b>	<b>185</b>
II.1	Key Modules in the SLAM Front-End	185
II.1.1	Odometry	186
II.1.2	Loop Closure	186
II.1.3	Priors and Unary Factors	187
II.2	Sensors and Factor Graphs	187
II.2.1	Selecting the Right Sensor for Your Application	188
II.2.2	Sensor Fusion	189
II.2.3	Calibration and Synchronization of Sensors	190
II.3	Evaluation	191
II.4	How to Read Part II?	192
<b>7</b>	<b>Visual SLAM</b>	<b>193</b>
7.1	Historical Background and Terminology	193
7.1.1	From Photogrammetry to Bundle Adjustment and Visual SLAM	193
7.1.2	Terminology	194
7.2	The Processing Pipeline of a Visual SLAM System	196
7.2.1	Visual Odometry Front-End	196
7.2.2	Mapping Back-End	196
7.2.3	Visual Place Recognition and Relocalization	197
7.2.4	Compute and Data Flow	197
7.3	Visual SLAM Fundamentals	197
7.3.1	Camera Model	197

	<i>Contents</i>	ix
7.3.2	Keypoints	200
7.3.3	Reprojection Error	204
7.3.4	Keypoint-Based Visual SLAM	205
7.3.5	Photometric Error and Direct Methods	207
7.3.6	Visual Place Recognition and Global Localization	207
7.3.7	Initialization	207
7.3.8	Map Representations	207
7.4	Further Considerations about Image Alignment and BA	207
7.4.1	Keypoint-based Image Alignment	208
7.4.2	Direct Image Alignment	209
7.4.3	Solving BA	211
7.4.4	Bundle Adjustment Revisited	213
7.5	Examples of Full Visual SLAM Systems	215
7.6	Real-time Dense Reconstruction	216
7.7	SLAM with Depth-sensing Cameras	217
7.8	Combining Vision with Other Modalities	219
7.8.1	Inertial Measurement Units (IMU)	219
7.8.2	GPS and WiFi for Global Localization	221
7.9	Further Readings & Recent Trends	222
<b>8</b>	<b>LiDAR SLAM</b>	<b>224</b>
8.1	LiDAR Sensing Preliminaries and Categorization	225
8.2	LiDAR Odometry	227
8.2.1	Foundations of Scan Registration	228
8.2.2	Common Components for LiDAR Odometry	231
8.2.3	Summary of LiDAR Odometry	236
8.3	LiDAR Place Recognition	236
8.3.1	Problem Definition	237
8.3.2	Methods for LiDAR Place Recognition	238
8.3.3	Summary of LiDAR Place Recognition	240
8.4	LiDAR SLAM	240
8.4.1	Structure of a LiDAR SLAM System	242
8.4.2	Pose-graph Optimization and Map Update	243
8.4.3	Multi-robot and Multi-session LiDAR SLAM	245
8.5	Further Readings & Recent Trends	248
<b>9</b>	<b>Radar SLAM</b>	<b>250</b>
9.1	Introduction to Radar	250
9.1.1	Sensor Types	250
9.1.2	Radar Sensing Principles	252
9.1.3	Challenges to Radar Applications	258
9.1.4	Radar Filtering	259
9.2	Radar Odometry	261

9.2.1	Doppler Odometry	262
9.2.2	Direct Odometry	265
9.2.3	Feature-based Odometry	265
9.2.4	Registration-based Odometry	266
9.2.5	Motion Compensation	268
9.3	Radar Place Recognition	268
9.3.1	Unique Challenges in Radar Place Recognition	269
9.3.2	Learning-based Radar Place Recognition	270
9.3.3	Descriptor-based Radar Place Recognition	271
9.4	Radar SLAM	273
9.4.1	Map Representations	273
9.4.2	Radar SLAM Pipelines	274
9.4.3	Multi-modality in Radar SLAM	277
9.5	Radar Datasets	279
9.6	Further Readings & Recent Trends	280
<b>10</b>	<b>Event-based SLAM</b>	<b>282</b>
10.1	Sensor Description	282
10.1.1	Working principle	282
10.1.2	Advantages of Event Cameras	285
10.1.3	Current Devices and Trends	285
10.2	Challenges and Applications	286
10.3	Overview and Taxonomy of Event-based SLAM Methods	287
10.4	Front-end of an Event-based SLAM System	289
10.4.1	Pre-processing and Event Representations	290
10.4.2	Indirect Methods	291
10.4.3	Direct Methods	291
10.4.4	Model-based and Learning-based Methods	292
10.5	Back-end of an Event-based SLAM System	293
10.6	State-of-the-Art Systems	294
10.7	Datasets, Simulators, and Benchmarks	294
10.7.1	Simulators	295
10.7.2	Datasets and Benchmarks	298
10.7.3	Metrics	301
10.8	Further Readings & Recent Trends	302
<b>11</b>	<b>Inertial Odometry for SLAM</b>	<b>304</b>
11.1	Basics of Inertial Sensing and Navigation	304
11.1.1	Sensing Principles and Measurement Models	305
11.1.2	Initial Alignment	307
11.2	IMU Preintegration and Factor Graphs	308
11.2.1	Motion Integration	308

11.2.2	inertial measurement unit (IMU) Preintegration on Manifold	311
11.2.3	Advanced Preintegration Techniques	316
11.3	Observability of Aided Inertial Navigation	319
11.3.1	Linearized Measurement Models	320
11.3.2	Observability Analysis	324
11.3.3	Degenerate Motions	325
11.4	Visual-Inertial Odometry and Practical Considerations	326
11.4.1	Visual-Inertial Odometry	326
11.4.2	Extrinsic Calibration	329
11.4.3	Temporal Synchronization	329
11.5	Further Readings & Recent Trends	330
12	<b>Leg Odometry for SLAM</b>	333
12.1	Historical Background and Preliminaries	333
12.1.1	Historical Background	334
12.1.2	Reference Frames	335
12.1.3	State Definition	335
12.1.4	Legged Robot Kinematics	336
12.1.5	Legged Robot Dynamics	338
12.1.6	Joint Sensing	339
12.2	Motion Estimation	342
12.2.1	Relative Pose Estimation	343
12.2.2	Velocity Estimation	344
12.3	Contact Estimation	345
12.3.1	With Contact Sensors	346
12.3.2	With Force/Torque Sensors	347
12.3.3	With IMUs	347
12.3.4	From Joint Torque Sensing	347
12.4	Using Leg Odometry for State Estimation	348
12.4.1	Encoder Noise Propagation	348
12.4.2	Factor Graph Smoothing	349
12.4.3	Integration with Exteroceptive Sensors for SLAM	353
12.5	Open Challenges	353
12.5.1	Leg Deformation	353
12.5.2	Non-rigid Contacts and Slippage	354
12.6	Further Readings & Recent Trends	355
	<b>PART THREE FROM SLAM TO SPATIAL AI</b>	359
III	<b>Prelude</b>	361
III.1	Spatial Artificial Intelligence	361

III.2	Spatial AI Applications	362
III.2.1	AR/VR Glasses	362
III.2.2	AI Glasses	363
III.2.3	Humanoid Robots	363
III.2.4	Self-Driving Vehicles	363
III.2.5	Drones and Aerial Robotics	364
III.2.6	Industrial Automation and Warehousing	364
III.2.7	Emergency Response	364
III.2.8	Healthcare and Assistive Technologies	364
III.3	How to Read Part III?	365
<b>13</b>	<b>Boosting SLAM with Deep Learning</b>	366
13.1	Deep Learning for Depth and Camera Pose	368
13.1.1	Deep Learning for Depth Prediction	369
13.1.2	Deep Learning for Camera Pose Prediction	370
13.1.3	Unsupervised Learning of Depth and Camera Pose	371
13.2	Deep Learning for Feature Matching and Optical Flow	374
13.2.1	Learning Feature Detectors and Descriptors	374
13.2.2	Feature Matching	374
13.2.3	Optical Flow and RAFT	375
13.2.4	Optical Flow as Visual Measurements	378
13.2.5	Estimating Pose and Depth from Optical Flow	378
13.3	Differentiable Bundle Adjustment and DROID-SLAM	379
13.3.1	DROID-SLAM Architecture	380
13.3.2	DROID-SLAM Inference	382
13.3.3	Generalizing to Other Modalities	383
13.3.4	Deep Patch Visual Odometry (DPVO)	384
13.4	DuSt3R	386
13.4.1	A Network for Generalized Stereo Reconstruction	388
13.4.2	Regression Loss and Confidence-Aware Loss	389
13.4.3	Downstream Applications	389
13.4.4	Global Alignment	391
13.5	MASt3R	392
13.5.1	Matching Head	392
13.5.2	Matching Objective	392
13.6	Extending MASt3R to structure from motion (SFM) and SLAM	393
13.6.1	MASt3R-SfM	393
13.6.2	MUSt3R	394
13.6.3	MASt3R-SLAM	394
13.7	Further Readings & Recent Trends	395
<b>14</b>	<b>Map Representations with Differentiable Volume Rendering</b>	397
14.1	3D Scene Representation and Differentiable Rendering	398

	<i>Contents</i>	xiii
14.1.1	Learnable 3D Representations	398
14.1.2	Differentiable Rendering	399
14.1.3	3D Scene Representation with Differentiable Rendering	399
14.2	Neural Radiance Fields (neural radiance field (NeRF))	401
14.2.1	Method Overview	402
14.2.2	Data Structures in NeRF	403
14.2.3	Neural Fields	404
14.2.4	NeRF/Neural Fields for Visual SLAM	405
14.3	3D Gaussian Splatting	409
14.3.1	Method Overview	409
14.3.2	Applications of 3D Gaussian Splatting	410
14.3.3	3D Gaussian Splatting for SLAM	411
14.4	Further Readings & Recent Trends	414
<b>15</b>	<b>Dynamic and Deformable SLAM</b>	<b>417</b>
15.1	Characterizing the Dynamic SLAM Problem	418
15.1.1	Characterizing Dynamics	419
15.1.2	State Estimation vs. Scene Representation	422
15.1.3	Online vs. Offline Methods	422
15.2	Short-term Dynamics and Dynamic SLAM	423
15.2.1	Dynamic Object Removal	424
15.2.2	Dynamic Object Tracking	427
15.2.3	Dense Dynamic SLAM	428
15.3	Long-term Dynamic and Lifelong SLAM	432
15.3.1	Lifelong SLAM for Localization	433
15.3.2	Map Cleaning and Change Detection	435
15.3.3	Change-aware SLAM	441
15.3.4	Temporal Scene Understanding	444
15.4	Deformable SLAM	446
15.4.1	Non-Rigid Structure from Motion (NRSfM) vs. Deformable SLAM	446
15.4.2	Deformable SLAM with Depth Information	449
15.4.3	Pipeline for Deformable SLAM using Monocular Cameras	450
15.4.4	Initialization and Map Extension	451
15.5	Further Readings & Recent Trends	451
<b>16</b>	<b>Metric-Semantic SLAM</b>	<b>454</b>
16.1	From Traditional SLAM to Metric-Semantic SLAM	455
16.2	Sparse Metric-Semantic Representations	456
16.2.1	Object Representation and Factor Graph Modeling	456
16.2.2	Hybrid Solvers for Sparse Metric-Semantic SLAM	466
16.3	Dense Metric-Semantic Representations	471

16.3.1	Point-based and Surfel-based Metric-Semantic SLAM	471
16.3.2	Voxel-based Metric-Semantic SLAM	472
16.3.3	Mesh-based Metric-Semantic SLAM	479
16.4	Hierarchical Metric-Semantic Representations and 3D Scene Graphs	482
16.4.1	Hierarchical Representations and Symbol Grounding	483
16.4.2	3D Scene Graphs	486
16.5	Further Readings & Recent Trends	487
<b>17</b>	<b>Towards Open-World Spatial AI</b>	<b>490</b>
17.1	Background and Terminology	491
17.1.1	A Brief History of Foundation Models	491
17.1.2	Terminology and Scope	494
17.2	Foundation Models for Spatial AI	495
17.2.1	Feature-Based Foundation Models	496
17.2.2	Generative Foundation Models	497
17.2.3	Class-Agnostic Image Segmentation	499
17.3	Open-World Mapping	499
17.3.1	Dense Representations	500
17.3.2	Object Maps and 3D Scene Graphs	504
17.3.3	Implicit Functions	510
17.3.4	Task-Driven Representations	511
17.4	Further Readings & Recent Trends	514
17.4.1	Grounding Foundation Models with Maps	514
17.4.2	Revisiting the Question of the Need for Maps	515
17.4.3	A Foundation Model for Robotics?	516
17.4.4	Concluding Remarks	519
<b>18</b>	<b>The Computational Structure of Spatial AI Systems</b>	<b>521</b>
18.1	From SLAM to Spatial AI	521
18.1.1	Intelligent Embodied Devices Need Spatial AI	521
18.1.2	Scene Representation and World Models	522
18.1.3	SLAM is Evolving into Spatial AI	524
18.2	Overall Computational Structure	525
18.3	State Estimation and Machine Learning in Spatial AI	526
18.4	The Future Landscape of Processor and Sensor Hardware	528
18.4.1	Processors	528
18.4.2	Sensors	531
18.5	Mapping Spatial AI Graphs to Hardware	532
18.5.1	World Model Processing	534
18.5.2	Real-Time Loop	537
18.5.3	Processing Close to the Image Plane	538

<i>Contents</i>	xv
18.6 Convergent Distributed Computation with Gaussian Belief Propagation	540
18.7 Continual Learning within Factor Graphs	542
18.8 Performance Metrics	545
18.9 Further Readings & Recent Trends	547
<i>Epilogue</i>	548
 <i>References</i>	551
<i>Author index</i>	645
<i>Subject index</i>	646

## Preface

We are proud to present this handbook on SLAM after over two years of effort. Our vision was to produce a primer that could be handed to upper-year undergraduates or new graduate students that summarizes the current state of the art. We have attempted to produce a unified document where chapters build on one another. This was quite challenging with over 50 authors simultaneously writing but we are delighted with the result: a book by the community, for the community.

The term SLAM itself is about 40 years old, but the idea of building a map while localizing on it goes back much further. In places throughout the handbook we look at this historical path, but we are mainly focused on the present and the future of SLAM here. Accordingly, the handbook is divided into three parts:

**Part I:** We introduce the foundations of SLAM, focusing on the estimation-theoretic machinery of the back-end and the map representations it produces. We cover factor graphs, manifold state representation, handling outliers, differentiable optimization, dense mapping, and advanced solvers.

**Part II:** Here we study SLAM in practice and focus on the characteristics and integration of various sensor modalities used in SLAM, including RGB cameras, LiDARs, event cameras, radars, IMUs, and robot kinematics. We explain how these sensors contribute to odometry, loop closure, and other factors, and how they can be calibrated, synchronized, fused, and evaluated within a factor-graph framework.

**Part III:** We look forward to emerging trends in SLAM and Spatial AI, exploring deep learning integration, novel map representations, and operation in dynamic or deformable environments. We also consider semantic reasoning, language grounding, and future computational architectures for distributed spatial perception.

We would like to thank all the authors for their invaluable contributions. They are all experts in their respective disciplines and working with them has been an honor and a pleasure. We would also very much like to thank Wolfram Burgard, Dieter Fox, and Sebastian Thrun, the authors of the famous *Probabilistic Robotics* book for writing the foreword to this handbook; we are certainly standing on the shoulders of giants. We are indebted to Dongjae Lee, who helped immensely with the nuts and

bolts of producing such a large manuscript. We thank Oussama Khatib, co-editor of the *Handbook of Robotics*, for his seasoned advice on writing this handbook. We are especially grateful to Frank Park for his leadership in proposing a SLAM handbook two years ago; he was there from the start. Thanks also to Cambridge University Press for supporting this ambitious and hopefully impactful project.

It has been a wonderful journey bringing this handbook to fruition. We hope you enjoy reading these pages and much as we enjoyed bringing them together.

Luca Carlone, Ayoung Kim,  
Timothy Barfoot, Daniel Cremers, and Frank Dellaert



## Notation

### – GENERAL NOTATION –

$a$	This font is used for real scalars
$\mathbf{a}$	This font is used for real column vectors
$\mathbf{A}$	This font is used for real matrices
$\mathcal{A}$	This font is used for sets
$\mathbf{I}$	The identity matrix
$\mathbf{0}$	The zero matrix
$\mathbf{A}^\top$	The transpose of matrix $\mathbf{A}$
$\mathbb{R}^{M \times N}$	The vector space of real $M \times N$ matrices
$p(\mathbf{a})$	The probability density of $\mathbf{a}$
$p(\mathbf{a} \mathbf{b})$	The probability density of $\mathbf{a}$ given $\mathbf{b}$
$p(\mathbf{a}; \mathbf{b})$	The probability density of $\mathbf{a}$ parametrized by $\mathbf{b}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian probability density with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\mathcal{GP}(\boldsymbol{\mu}(t), \mathcal{K}(t, t'))$	Gaussian process with mean function, $\boldsymbol{\mu}(t)$ , and covariance function, $\mathcal{K}(t, t')$
$(\hat{\cdot})$	A posterior (estimated) quantity
$(\cdot)$	A prior quantity
$(\cdot)_k$	The value of a quantity at timestep $k$
$(\cdot)_{k_1:k_2}$	The set of values of a quantity from timestep $k_1$ to timestep $k_2$ , inclusive
$\ \cdot\ _1$	L1 norm $\ \mathbf{x}\ _1 = \sum  x_i $
$\ \cdot\ _2$	L2 norm $\ \mathbf{x}\ _2 = \sqrt{\sum x_i^2}$

## – 3D GEOMETRY NOTATION –

$\mathcal{F}^a$	A reference frame in three dimensions
$\mathbf{v}^a$	The coordinates of a vector in frame $\mathcal{F}^a$
$\mathbf{R}_a^b$	A $3 \times 3$ rotation matrix (member of SO(3)) that takes points expressed in $\mathcal{F}^a$ and re-expresses them in (purely rotated) $\mathcal{F}^b$ : $\mathbf{v}^b = \mathbf{R}_a^b \mathbf{v}^a$
$\tilde{\mathbf{v}}_a^b$	The three-dimensional position of the origin of frame $\mathcal{F}^a$ expressed in $\mathcal{F}^b$
$\tilde{\mathbf{v}}^a = \begin{bmatrix} \mathbf{v}^a \\ 1 \end{bmatrix}$	A $4 \times 1$ homogeneous point expressed in $\mathcal{F}^a$
$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}$	A $4 \times 4$ transformation matrix (member of SE(3)) that takes homogeneous points expressed in $\mathcal{F}^a$ and re-expresses them in (rotated and translated) $\mathcal{F}^b$ : $\tilde{\mathbf{v}}^b = \mathbf{T}_a^b \tilde{\mathbf{v}}^a$
SO(3)	The special orthogonal group, a matrix Lie group used to represent 3D rotations
so(3)	The Lie algebra associated with SO(3)
SE(3)	The special Euclidean group, a matrix Lie group used to represent 3D poses
se(3)	The Lie algebra associated with SE(3)
$(\cdot)^\wedge$	An operator mapping a vector in $\mathbb{R}^3$ (resp. $\mathbb{R}^6$ ) to an element of the Lie algebra for 3D rotations (resp. 3D poses); implements the cross product for three-dimensional quantities, <i>i.e.</i> , for two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$ , $\mathbf{u}^\wedge \mathbf{v} = \mathbf{u} \times \mathbf{v}$
$(\cdot)^\vee$	An operator mapping an element of the Lie algebra for 3D rotations (resp. 3D poses) to a vector in $\mathbb{R}^3$ (resp. $\mathbb{R}^6$ )

# **PART ONE**

## FOUNDATIONS OF SLAM



# I

## Prelude

Luca Carlone, Ayoung Kim, Frank Dellaert, Timothy Barfoot, and Daniel Cremers

This chapter introduces the Simultaneous Localization and Mapping (SLAM) problem, presents the modules that form a typical SLAM system, and explains the role of SLAM in the architecture of an autonomous system. The chapter also provides a short historical perspective of the topic and discusses how the traditional notion of SLAM is evolving to fully leverage new technological trends and opportunities. The ultimate goal of the chapter is to introduce basic terminology and motivations, and to describe the scope and structure of this handbook.

### I.1 What is SLAM?

A necessary prerequisite for a robot to operate safely and effectively in an unknown environment is to form an internal representation of its surroundings. This representation can be used to support obstacle avoidance, low-level control, planning, and, more generally, the decision-making processes required for the robot to complete the task it has been assigned. The execution of simple tasks (*e.g.*, following a lane, or maintaining a certain distance to an object in front of the robot) may only require tracking entities of interest in the sensor data streams, while complex tasks (*e.g.*, large-scale navigation or mobile manipulation) require building and maintaining a *persistent representation* (a map) of the environment. Such a map describes the presence of obstacles, objects, and other entities of interest, and their relative location with respect to the robot’s pose (position and orientation). For instance, the map might be used instruct the robot to reach a location of interest, to grasp a certain object, or to support the exploration of an initially unknown environment.

For a robot operating in an initially unknown environment, the problem of building a map of the environment, while concurrently estimating its pose with respect to that map, is referred to as *SLAM*. Figure I.1 provides some real-world examples of SLAM in action. SLAM reduces to *localization* if the map is given, in which case the robot only has to estimate its pose with respect to the map. On the other hand, SLAM reduces to *mapping* if the pose of the robot is already known, for instance when an absolute positioning system is used (*e.g.*, differential GPS or motion cap-

*SLAM, localization, mapping*

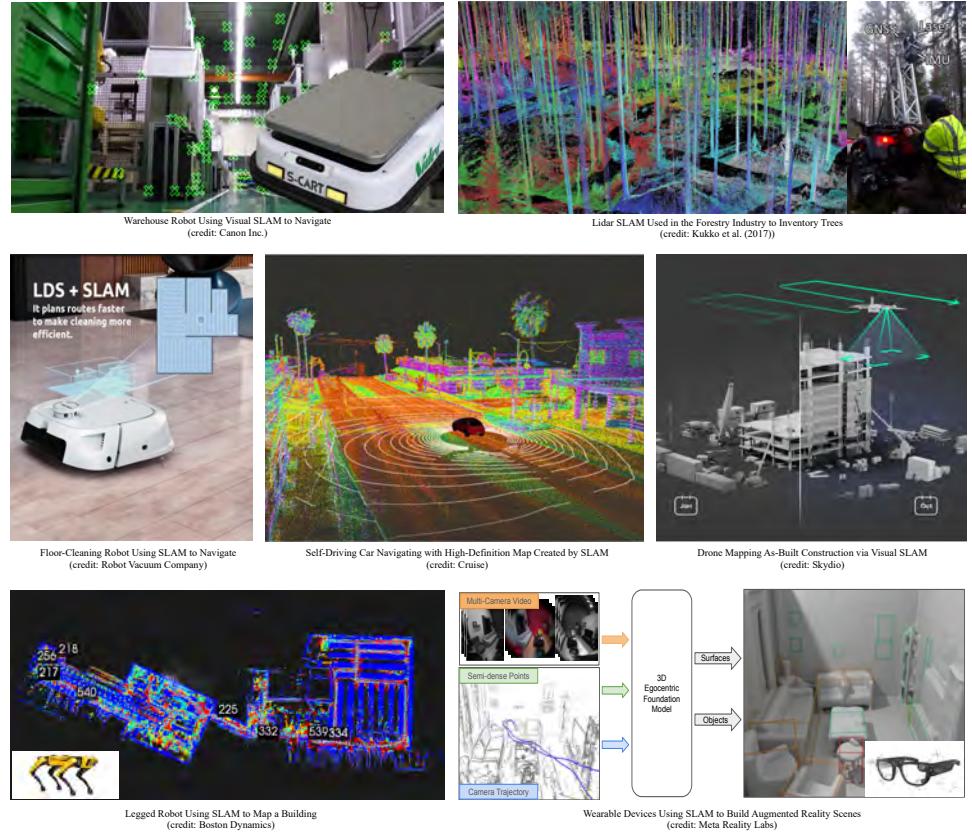


Figure I.1 SLAM is rapidly becoming an enabling technology in a wide array of applications including warehouse robotics, forest inventories [611], floor-cleaning, self-driving cars, drones surveillance, legged-robot mapping, and augmented reality to name only a few. (©2017 Elsevier)

ture), in which case the robot only needs to model its surroundings using its sensor data.

The central role of SLAM in robotics research is due to the fact that robot poses are rarely known in practical applications. Differential GPS and motion capture systems are expensive and restricted to small areas, hence being unsuitable for large-scale robot deployments. Consumer-grade GPS is much more broadly available, but its accuracy (with errors typically in the order of meters) and its availability (which is limited to outdoor areas with line-of-sight to satellites) often makes it unsuitable as a source of localization; the consumer-grade GPS —when available— is typically used as an additional source of information for SLAM, rather than a replacement for the localization aspects of SLAM.

Similarly, in many robotics applications, the robot will not typically have access

to a prior map, hence it needs to perform SLAM rather than localization. Indeed, in certain applications, building a map is actually the *goal* of the robot deployment; for instance, when robots are used to support disaster response and search-and-rescue operations, they might be deployed to construct a map of the disaster site to help first-responders. In other cases, the map might be stale or not have enough detail. For instance, a domestic robot might have access to the floor plan of the apartment it has to operate in, but such a floor plan may not describe the furniture and objects actually present in the environment, nor the fact that these elements can be rearranged from day to day. In a similar manner, Mars exploration rovers have access to low-resolution satellite maps of the Martian surface, but they still need to perform local mapping to guide obstacle avoidance and motion planning.

The importance of the SLAM problem motivates the large amount of attention this topic has received, both within the research community and from practitioners interested in using SLAM technologies across multiple application domains from robotics, to virtual and augmented reality. At the same time, SLAM remains an exciting area of research, with many open problems and new opportunities.

## I.2 Anatomy of a Modern SLAM System

The ultimate goal of SLAM is to infer a map representation and robot poses (*i.e.*, the robot trajectory) from sensor data, including data from *proprioceptive* sensors (*e.g.*, wheel odometry or inertial measurement unit, IMU) and *exteroceptive* sensors (*e.g.*, cameras, light detection and ranging (LiDAR), radars). In mathematical terms this can be understood as an *inverse problem*: given a set of measurements, the goal is to determine a model of the world (the map) and a set of robot poses (trajectory) that could have produced those measurements. There exist two alternative strategies to solve the SLAM problem: indirect and direct methods.

The vast majority of SLAM methods prefers pre-processing the raw sensory data in order to extract “intermediate representations” that are compact and easier to describe mathematically. For instance, in visual SLAM, instead of using every pixel in a camera image, these *indirect* methods extract a few distinctive *2D point features* (or keypoints) and then only model the geometry of how these keypoints depend on the pose of the camera and the geometry of the scene. In contrast, rather than computing an intermediate abstraction, *direct* methods aim to compute localization and mapping *directly* from the raw sensory data. This categorization is prominent in visual SLAM but is not limited to it as we will see in Chapter 8 and Chapter 9. Both indirect and direct methods have their advantages and shortcomings.

Indirect methods are often faster and more memory efficient, since they merely process a small subset of keypoints for which the 3D location is determined. As a consequence, real-time capable systems for indirect visual SLAM were already available around the year 2000. To date, indirect methods are the preferred approach for real-time robot vision on platforms with limited compute. Moreover, once the

*proprioceptive and exteroceptive sensors*

*indirect methods*

*direct methods*

intermediate representation is determined, the subsequent computations are often mathematically simpler, making the resulting inference problems more tractable. In the case of visual SLAM, for example, once a set of corresponding points is identified across a set of images, the resulting problem of localization and mapping amounts to the classical bundle adjustment (BA) problem for which a multitude of powerful solvers and approximation methods exist.

In turn, direct methods have the potential to provide superior accuracy because they make use of all available input information. While the processing of all available input information (for example all pixels in each image) is computationally cumbersome and capturing the complex relationship between the quantities of interest (localization and mapping) and the raw input data (*e.g.*, the brightness of each pixel) may create additional non-convexities in the loss function used for estimation, there exist efficient approximation and inference strategies with first real-time capable methods for direct visual SLAM emerging in the 2010s. As we will see in Part II and III, the efficient processing of huge amounts of input data can be facilitated by using graphics processing units (GPUs) to parallelize computations.

In both direct and indirect methods the measurements are used to infer the robot pose and map representation. There is a well-established literature in estimation theory describing how we can infer quantities of interest (in our case, the robot poses and the surrounding map) from observations. This book particularly focuses on estimation theoretic tools —reviewed and tailored to the SLAM problem in Chapter 1 and Chapter 2— that have their foundations in probabilistic inference and that rephrase estimation in terms of solving optimization problems.

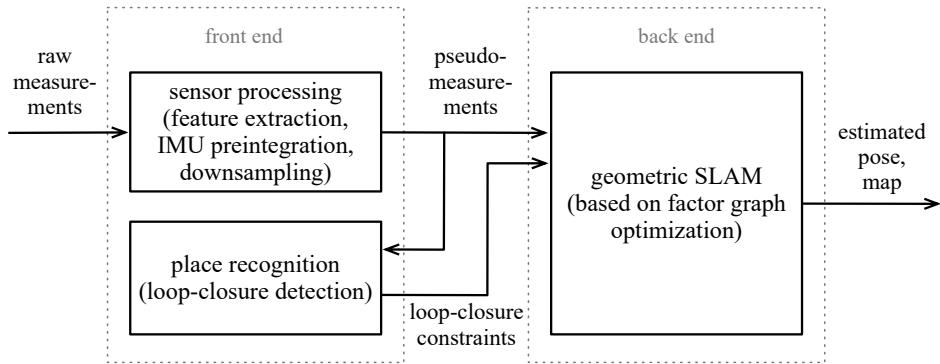


Figure I.2 Typical indirect methods for SLAM include a *front-end* (to process sensing data into a more manageable representation and to detect loop closures) and a *back-end* (to estimate the robot's pose and a geometric map). The back-end often has a number of helper modules aimed at helping with robustness, computational tractability, and map quality.

*SLAM front-end and back-end*

Indirect methods induce a natural split in common SLAM architectures (Figure

I.2): the raw sensor data is first passed to a set of algorithms (the *SLAM front-end*) in charge of extracting intermediate representations; then such intermediate representations are passed to an estimator (the *SLAM back-end*), that estimates the quantities of interest. The front-end is typically also in charge of building an *initial guess*: this is an initial estimate the back-end can use for iterative optimization, hence mitigating convergence issues due to non-convexity. Let us discuss a few examples to clarify the difference between the SLAM front-end and back-end.

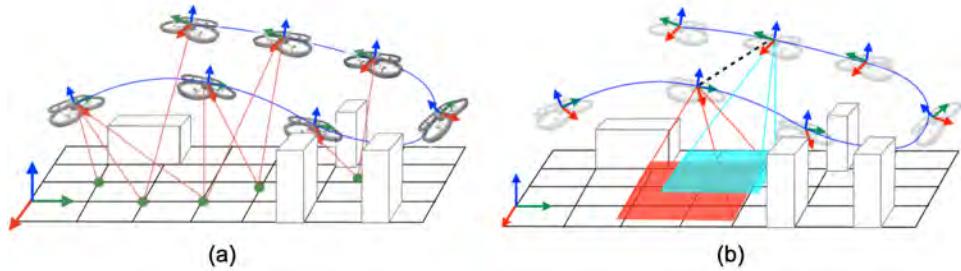


Figure I.3 (a) In landmark-based SLAM models, the front-end produces measurements to 3D landmarks and the back-end estimates the robot trajectory (as a set of poses) and landmark positions. (b) In pose-graph-based SLAM models, the front-end abstracts the raw sensor measurements in terms of odometry and loop closure measurements (these are typically relative pose measurements) and the back-end estimates the overall robot trajectory.

**Example I.1** (Visual SLAM: from pixels to landmarks) Visual SLAM uses camera images to estimate the robot trajectory and a sparse 3D point cloud map. The typical front-end of a visual SLAM system extracts 2D keypoints in each image and matches them across frames such that each group (a feature track) corresponds to re-observations of the same 3D point (a landmark) across different camera views. The front-end also computes rough estimates of the camera poses and 3D landmark positions by using computer vision techniques known as *minimal solvers*.<sup>1</sup> Then, the back-end is in charge of estimating (or refining) the unknown 3D position of the landmarks and the robot poses observing them by solving an optimization problem, known as *bundle adjustment*. This example leads to a landmark-based (of feature-based) SLAM model, visualized in Figure I.3(a). We will discuss visual SLAM at length in Chapter 7.

**Example I.2** (LiDAR SLAM: from scans to odometry and loop closures) LiDAR SLAM uses LiDAR scans to estimate the robot trajectory and a map. A common

<sup>1</sup> A more subtle point is that minimal solvers also allow pruning away a large portion of outliers, *i.e.*, incorrect detections of a landmark. This makes the job of the back-end easier, while still allowing it to remove any remaining outlier. We discuss outlier rejection and the related problem of data association in Chapter 3.

*pose-graph optimization  
and pose-graph-based  
SLAM*

front-end for LiDAR SLAM consists in using scan matching algorithms (*e.g.*, the Iterative Closest Point or ICP) to compute the relative pose between two LiDAR scans. In particular, the front-end will match scans taken at consecutive time instants to estimate the relative motion of the robot between them (the so called *odometry*) and will also match scans corresponding to multiple visits to the same place (the so called *loop closures*). Odometry and loop closure measurements are then passed to the back-end that optimizes the robot trajectory by solving an optimization problem, known as *pose-graph optimization (PGO)*. This example leads to a pose-graph-based SLAM model, visualized in Figure I.3(b). We discuss LiDAR SLAM in Chapter 8.

*landmarks, odometry, and  
loop closures*

The previous examples showcase three popular examples of “intermediate representations” (or pseudo-measurements) that are produced by the front-end and passed to the back-end (Figure I.2): landmark observations, odometry, and loop closures. In complex SLAM systems, these representations can be used in combination: for instance, in certain visual-SLAM systems one might extract keypoints corresponding to 3D landmarks, and further process them to compute relative poses corresponding to odometry and loop closures, and finally use a pose-graph-based back-end. The choice of the front-end/back-end split is about selecting a desired trade-off between computation and accuracy. Extracting simpler representations might lead to much faster back-end solvers (*e.g.*, performing pose-graph optimization is typically much faster than doing BA); but at the same time abstracting measurements induces approximation in how the measurements are modeled in the back-end, hence leading to small inaccuracies (*e.g.*, BA is typically more accurate than pose-graph optimization).

We remark that loop closures are a key aspect of SLAM. If we only use odometry for trajectory estimation, the resulting estimate—obtained by accumulating odometry motion estimates—is bound to drift over time, leading to severe distortion in the trajectory estimate. Revisiting already visited places is crucial to keep the trajectory estimation error bounded and obtain globally consistent maps. We also remark that loop closures are implicitly captured in landmark-based SLAM, where loop closures correspond to new observations of previously seen landmarks.

*multi-disciplinarity  
of SLAM*

We conclude this section by observing how SLAM research cuts across multiple disciplines. The SLAM front-end extracts features from raw sensor data, hence touching disciplines ranging from signal processing, geometry, 2D computer vision, and machine learning. The SLAM back-end performs estimation given measurements from the front-end, hence touching estimation theory, optimization, and applied mathematics. This variety of ideas and influences contribute to making SLAM a fascinating and multi-faceted problem.

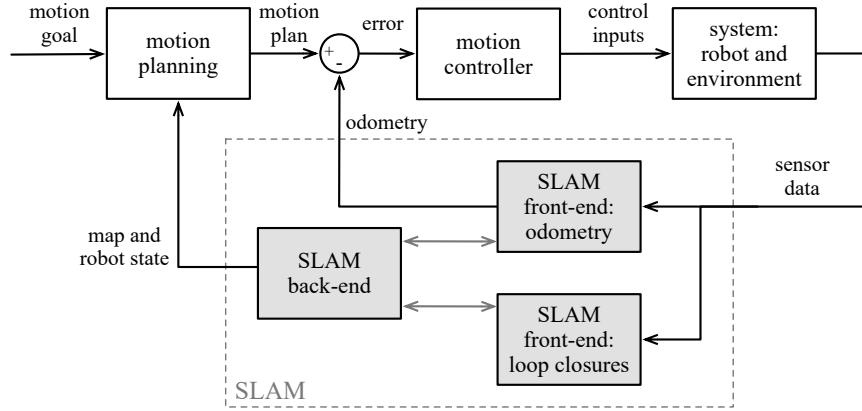


Figure I.4 SLAM plays an important role in the overall autonomy pipeline of a robot that interacts with the world, and provides necessary information for control and motion planning.

### I.3 The Role of SLAM in the Autonomy Architecture

The role of SLAM is to serve downstream tasks. For instance, the robot-pose estimate can be used to control the robot to follow a desired trajectory, while the map (in combination with the current robot pose) can be used for motion planning (Figure I.4). Here motion planning is used in a broad sense: while SLAM is typically used to build large-scale maps to support navigation tasks, it can also support building local 3D maps to enable manipulation and grasping.

While it would be tempting to think about SLAM as a monolithic system that takes sensor data in input and instantaneously outputs robot poses and map, the actual implementation of these systems and their integration in autonomy architectures is more complicated in practice. This is due to the fact that the robot needs to close different control and decision-making loops with different latency requirements. For instance, with reference to Figure I.4, the robot will need to close low-level control loops over its trajectory (this is the standard feedback control loop at the top-right of the figure), which might require relatively high rates and low-latency to be stable; for instance, a UAV flying at high speed might need the front-end to produce odometry estimates with a latency of a few milliseconds. On the other hand, closing the loop over motion planning (the outer loop in Figure I.4) can accommodate higher latencies, since global planning typically runs at lower rates; hence it might be acceptable for the back-end to provide global trajectory and map estimates with a latency of seconds. For these reasons, a typical implementation of a SLAM system involves multiple processes running in parallel and in a way that slower processes (*e.g.*, global pose and map optimization in the back-end) do not get in the way of faster processes (*e.g.*, odometry estimation).

*multi-threading in SLAM systems*

We also observe that the processes involved in a SLAM system have complex interactions (as emphasized by the bi-directional edges in Figure I.4): for instance, while the front-end feeds the odometry to the back-end, the back-end periodically applies global corrections to the odometric trajectory, which is then passed to the motion controller; similarly, while the front-end computes loop closures that are fed to the back-end, the back-end can also inform loop closure detection about plausible or implausible loop closure opportunities.

While the SLAM back-end might run at a slower pace, it is important to emphasize that it must remain *online*: it is desirable for the overall SLAM system to have a reasonable runtime that does not grow unbounded over time, and can be achieved on embedded robotics hardware as data streams are collected. These real-time constraints are vital for the robot to properly act in a complex environment, in particular with faster robots such as drones. This can be considered a main feature that historically differentiated SLAM from related problems in computer vision, such as *structure from motion (SFM)*: while SFM is also used to reconstruct the geometry of a 3D scene from camera images, it often relies on powerful computers (*e.g.*, a cluster of servers as in [13]), its runtime can be in the order of hours, and it is commonly applied to an unordered dataset of images. On the other hand, SLAM involves data causally collected by the robot over time as the robot explores the environment, and must run in seconds in the face of tight computational constraints. It is worth emphasizing that the boundaries between visual SLAM and SFM have become increasingly blurry, also thanks to work on online SFM in vision (going back to the early 2000s, *e.g.*, [523, 209]), and the use of SLAM methods for post-processing datasets offline. Therefore many researchers might use visual SLAM and SFM as interchangeable terms.

*SLAM and Structure from Motion*

### **I.3.1 Do We Really Need SLAM for Robotics?**

From our description above, SLAM feels like an intriguing but very challenging problem, ranging from its complex implementation, to the need of fast runtime on resource-constrained platforms. Therefore, a fair question to ask is whether we can develop complex autonomous robots that do *not* rely on SLAM. We refine this question into three sub-questions.

*SLAM and spatial memory*

**Q1. Do we need SLAM for any robotics task?** We started this section stating that SLAM is designed to support robotics tasks. Then a natural question is whether it is necessary for *any* robotics task. The answer is clearly: no. More reactive tasks, for instance keeping a target in sight, can be solved with simpler control strategies (*e.g.*, visual servoing).<sup>2</sup> Similarly, if the robot has to operate over small distances, relying on odometry estimates and local mapping might be acceptable. Moreover, if the environment the robot operates in has some infrastructure for

<sup>2</sup> One could argue that while not being strictly necessary for tracking, SLAM and odometry might still be helpful to increase its robustness, *e.g.*, when the target gets out of sight.

localization, then we may not need to solve SLAM. Nevertheless, SLAM seems an indispensable component for long-term robot operation in unstructured (*i.e.*, infrastructure-free) environments: long-term operation typically requires *memory* (*e.g.*, to go back to previously seen objects or find suitable collision-free paths), and map representations built from SLAM provide such a long-term memory.

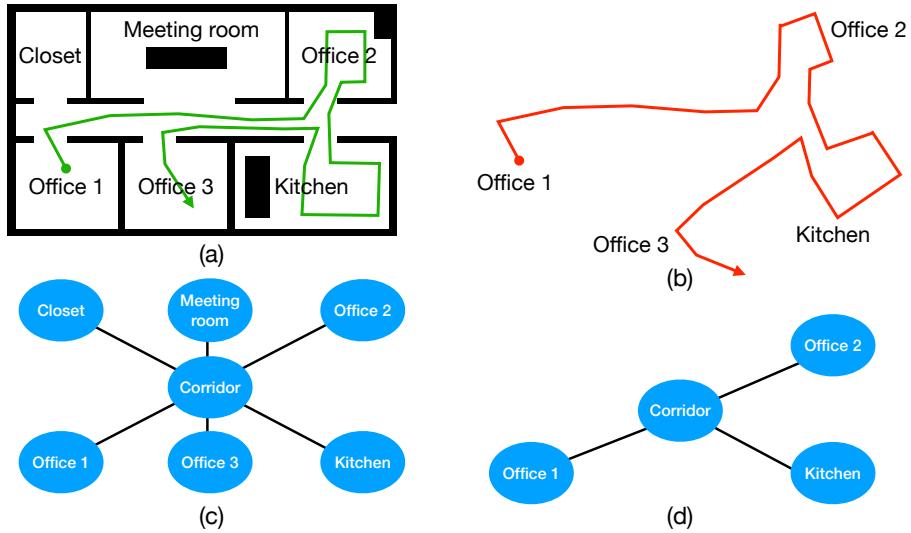


Figure I.5 (a) Our robot visits Office 1 in a building and then —after exploring other areas (including Office 2 and the Kitchen)— it visits Office 3, which is just next door from Office 1. Obstacles are shown in black and ground-truth trajectory is shown in green. (b) Odometric estimate of the trajectory, labeled with corresponding room labels. (c) Ground-truth topological map of the environment. (d) Estimated topological map in the presence of perceptual aliasing, causing the robot to think that Office 1 and 3 are the same room.

**Q2. Do we need globally consistent geometric maps for navigation?** A major focus in SLAM is to optimize the trajectory and map representations such that they are metrically accurate (or *globally consistent*) — this is precisely the role of the SLAM back-end. One might ask whether metric accuracy is actually needed. One alternative that comes to mind is to just use odometry to get locally consistent trajectory and map estimates; this circumvents the need for loop closures and back-end optimization. Unfortunately, due to its drift, odometry is unsuitable to support long-term operation: imagine that our robot visits Office 1 in a building and then, after exploring other areas of the building, it visits Office 3, which is just next door from Office 1 (see Figure I.5(a)). Using just odometry, the robot might be misled to conclude that Office 1 and Office 3 are quite far from each other (due to the odometry drift), hence being unable to realize there is a short path connecting the two offices (Figure I.5(b)). A slightly more sophisticated alternative is to

metric maps vs. topological maps

*Perceptual aliasing*

build a *topological map* instead. A topological map can be thought of as a graph where nodes are places the robot visited and edges represent traversability between the places connected by each edge (Figure I.5(c)). The difference with the *metric* SLAM lens we adopt in this handbook is that nodes and edges in a topological map do not carry metric information (distances, bearing, positions), hence they do not require any optimization: one can simply add edges to a topological map when the robot traveled between two places (odometry) or when a place recognition module recognizes the places to overlap (loop closures). While this seems a perfectly reasonable approach, the main issue is that place recognition techniques are not perfect and, more fundamentally, two different places might look similar (a phenomenon known as *perceptual aliasing*). Therefore, going back to our example above, if Office 1 and Office 3 look very similar, a purely topological approach might be misled to think there is a single office instead (Figure I.5(d)). On the other hand, metric SLAM approaches can use geometric information to conclude that the two offices are indeed two different rooms, by giving the user access to a more powerful set of tools to decide whether place recognition results are correct and if two observations correspond to the same place; we will discuss these tools at length in Chapter 3.

**Q3. Do we need maps?** SLAM builds a map that can be directly queried, inspected, and visualized. As we will see in Chapter 5, there are many ways to represent a map, including 3D point clouds, voxels, meshes, neural radiance fields, and others. On the other hand, one might take a completely different approach: in order for the robot to execute a task, the robot might be trained to translate raw sensor data directly to actions (*e.g.*, using Reinforcement Learning), hence circumventing the need to build a map. In such an approach, the neural network trained from sensor data to actions will arguably create an internal representation, but such an internal representation cannot be directly queried, inspected, or visualized. While the jury is still out on whether maps are indeed necessary, there is some initial evidence that using maps as an intermediate representation is at least beneficial in completing many visual tasks for robotics [909, 1249]. Moreover, maps have the benefit of being useful across a wide variety of tasks, while a representation that is fully learned in the context of a single task might not be able to support new unseen tasks. Finally, we observe that there are several applications where the *goal* is to have a map that can be inspected. This is the case in search-and-rescue robotics applications where it is desirable to provide a map to help first-responders. Moreover, it is the case for several applications beyond robotics (*e.g.*, real-estate planning and visualization, construction monitoring, virtual and augmented reality), where the goal is for a human to inspect or visualize the map.

## I.4 Past, Present, and Future of SLAM, and Scope of this Handbook

The design of algorithms for spatial reasoning has been at the center-stage of robotics and computer vision research since their inception. At the same time, SLAM research keeps evolving and expanding to novel tools and problems.

### I.4.1 Short History and Scope of this Handbook

As discussed across the various chapters of this book, SLAM has multiple facets. As a consequence, its history is also multi-faceted with origins that can be traced back across different scientific communities.

Creating maps of the world from observations and measurements is among the oldest challenges in history and leads to the fields of *geodesy* (the science measuring properties of the Earth) and *surveying*. There are many pioneers who contributed to this field. Carl Friedrich Gauss triangulated the Kingdom of Hannover in the years 1821–1825. Sir George Everest served as Surveyor General of India 1830–1843 in the Great Trigonometric Survey, efforts for which he was honored by having the world’s highest mountain named after him. In 1856, Carl Maximilian von Bauernfeind published a standard book on “Elements of Surveying” [60]. He subsequently founded the Technical University of Munich in 1868 with a central focus on establishing geodesy as a scientific discipline. André-Louis Cholesky developed the well-known Cholesky matrix decomposition while surveying Crete and North Africa before the First World War.

The problem of visual SLAM is also closely related to the field of photogrammetry and the problems of Structure from Motion, whose origins can be traced back to the 19th century. We discuss this further in Chapter 7.

In robotics, the origin of SLAM is typically traced back to the seminal work of Smith and Chessman [1015] and Durrant-Whyte [293], as well as the parallel work by Crowley [233] and Chatila and Laumond [172]. The acronym SLAM was coined in 1995, as part of the survey paper [294]. These early works developed two fundamental insights. The first insight is that to avoid drift in unknown environments, one needs to simultaneously estimate the robot poses and the position of fixed external entities (*e.g.*, landmarks). The second insight is that existing tools from estimation theory, and in particular the celebrated Extended Kalman Filter (EKF), could be used to perform estimation over an extended state describing the robot poses and the landmark positions, leading to a family of *EKF-SLAM* approaches.

EKF-SLAM approaches have been extremely popular but face three main issues in practice. The first is that they are sensitive to outliers and data association errors. These errors may result from failures of place recognition or object detection, where the robot believes it is observing a given object or place, but it is actually observing a different (but possibly similarly looking) one. If these spurious measurements are not properly handled, EKF-SLAM produces grossly incorrect estimates.

*EKF-SLAM*

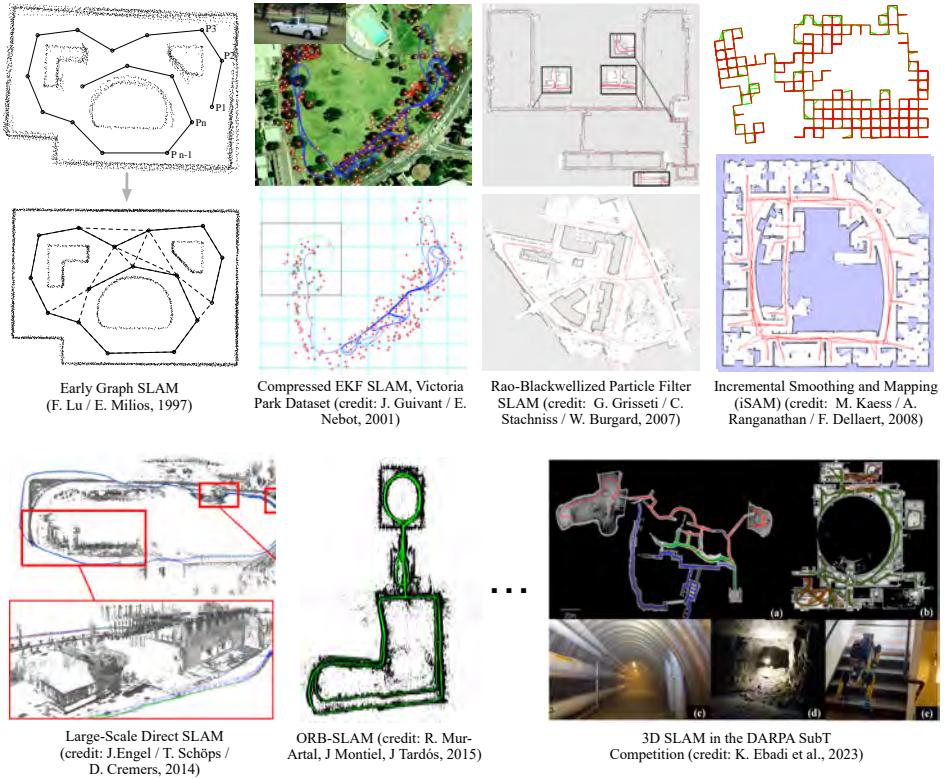


Figure I.6 The history of SLAM is filled with numerous advances that have led to modern SLAM systems capable of localizing and mapping robots in challenging real-world environments. This image shows a selection of representative highlights. Figures from [698] (©1997 Springer), [413] (©2001 IEEE), [399] (©2007 IEEE), and [784] (©2015 IEEE).

The second issue is related to the fact that EKF relies on linearization of the equations describing the motion of the robot and sensor observations. In practice, the linearization point is typically built from odometry and when the latter drifts, the linearized system might be a poor approximation of the original nonlinear system. This leads EKF-SLAM to diverge when odometry accumulates substantial drift. The third problem is about computational complexity: a naive implementation of the Kalman Filter leads to a computational complexity that grows quadratically in the number of state variables, due to the need to manipulate a dense covariance matrix. In a landmark-based SLAM problem it is not uncommon to have thousands of landmark, which makes the naive approach prohibitive to run in real-time.

As a response to these issues, in the early 2000s, the community started focusing on *particle-filter-based approaches* [768, 1010, 399], which model the robot trajectory using a set of hypothesis (or *particles*), building on the theory of particle filtering

in estimation theory.<sup>3</sup> When used in combination with landmark-based maps, these models allowed using a large number of landmarks (breaking through the quadratic complexity of the EKF); moreover, they allowed to more easily estimate dense map models, such as 2D occupancy grid maps. Also, these approaches did not rely on linearization and were less sensitive to outliers and incorrect data association. However, they still exhibited a trade-off between computation and accuracy: obtaining accurate trajectories and maps requires using many particles (in the thousands) but the more particles, the more computation. In particular, for a finite amount of particles, a particle filter may still diverge when none of the sampled particles are near the real trajectory of the robot (an issue known as *particle depletion*); this issue is exacerbated in 3D problems where one needs many particles to cover potential 3D poses of the robot.

Between 2005 and 2015, a key insight pushed to the spotlight an alternative approach to SLAM. The insight is that while the covariance matrix appearing in the EKF is dense, its inverse (the so called *Information Matrix*) is very sparse and has a very predictable sparsity pattern when past robot poses are retained in the estimation [313]; this allows designing filtering algorithms that have close-to-linear complexity, as opposed to the quadratic complexity of the EKF. While this insight was initially applied to EKF-like approaches, such as EIF, it also paved the way for *optimization-based approaches*. Optimization-based approaches were first proposed in the early days of SLAM [698], but then disregarded as too slow to be practical. The sparsity structure mentioned above allowed rethinking these optimization methods and making them more scalable and solvable in online fashion [249, 531].<sup>4</sup> This new wave can be interpreted as a shift toward yet another estimation framework: *maximum likelihood* and *maximum a posteriori* estimation. These frameworks rephrase estimation problems in terms of optimization, while describing the structure of the problem in terms of a probabilistic graphical model, or, specifically, a *factor graph*. The resulting factor-graph-based approach to SLAM is still the dominant paradigm today, and has also shaped the way the community thinks about related problems, such as visual and visual-inertial odometry. The optimization lens is a powerful one and allows a much deeper theoretical analysis than previously possible (see Chapter 6). Moreover, it is fairly easy to show that the EKF (with suitable linearization points) can be understood as a single iteration of a nonlinear optimization solver, hence making the optimization lens strictly more powerful than its filtering-based counterpart. Finally, the optimization-based perspective appears more suitable for recent extensions of SLAM (described in the next section and Part III of this handbook), where one wants to estimate both continuous variables

*Factor graphs*

<sup>3</sup> The resulting algorithms are known with different names in different communities, *e.g.*, Sampling/Importance Re-sampling, Monte-Carlo filter Condensation algorithm, Survival of the fittest algorithm, and others.

<sup>4</sup> More details are in Chapter 1.

(describing the scene geometry) and discrete variables (describing semantic aspects of the scene).

This short history review stops at 2015, while the goal of Part III of this handbook is to discuss more modern trends, including those triggered by the “deep learning revolution”, which started around 2012 and slowly permeated to robotics. We also remark that the short history above mostly gravitates around what we called the SLAM back-end (essentially, the estimation engine), while the development of the SLAM front-end traces back to work done across multiple communities, including computer vision, signal processing, and machine learning.

As a result of the considerations mentioned above, this handbook will primarily focus on the factor-graph-based formulation of SLAM. This is a decision about scope and does not detract from the value of ongoing works using other technical tools. For instance, at the time of writing of this handbook, EKF-based tools are still popular for visual-inertial odometry applications (building on the seminal work from Mourikis and Roumeliotis [774]), and novel estimation formulations have been developed, including invariant [56] and equivariant filters [331], as well as alternative formulations based on random finite sets [780].

#### ***I.4.2 From SLAM to Spatial AI***

SLAM essentially focuses on estimating geometric properties of the environment (and the robot). For instance, the SLAM map carries information about obstacles in the environment, distances and traversable paths between two locations, or geometric coordinates of distinctive landmarks. In this sense, SLAM is useful as a representation for the robot to understand and execute commands such as “robot: go to position  $[x, y, z]$ ”, where  $[x, y, z]$  are the coordinates (in the map frame) of a place or object the robot has to reach. However, specifying goals in terms of coordinates is not suitable for non-expert human users and it is definitely not the way we interact or specify goals for humans. Therefore, it would be desirable for the next generation of robots to understand and execute high-level commands specified in natural language, such as “robot: pick up the clothes in the bathroom, and take them to the laundry room”. Parsing these instructions requires the robot to understand both geometry (*e.g.*, where is the bathroom) and semantics (*e.g.*, what is a bathroom or laundry room, which objects are clothes) of the environment.

This realization has recently pushed the research community to think about SLAM as an integrated component of a broader *spatial perception* system, that simultaneously reasons about geometric, semantic, and possibly physical aspects of the scene, in order to build a multi-faceted map representation (a “world model”), that enables the robot to understand and execute complex instructions. The resulting *Spatial AI* algorithms and systems have the potential to increase robot autonomy and have rapidly progressed over the last decade. Intuitively, one can think that Spatial AI has SLAM as a submodule (to handle the geometric reason-

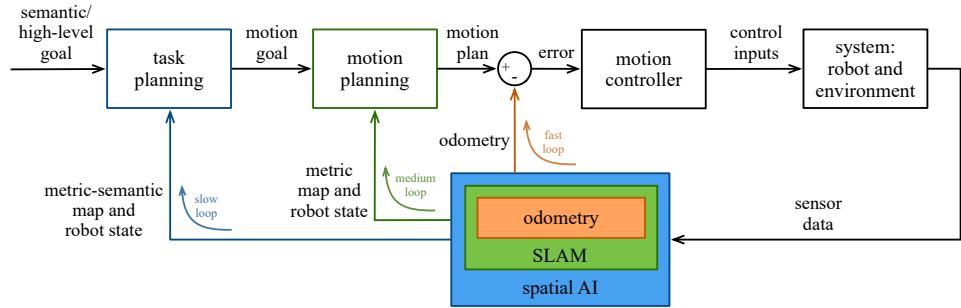


Figure I.7 Spatial AI (or spatial perception) extends the geometric reasoning capabilities of SLAM to also perform semantic and physical reasoning. While the SLAM block is informed by odometry and provides a geometric understanding of the scene, the Spatial AI block is informed by the SLAM results and adds a scene understanding component, spanning semantics, affordances, dynamics, and more. This allows closing the loop over higher-level decision making modules, such as task planning, and allows the user to specify higher-level goals the robot has to achieve.

ing part), but provides extra semantic reasoning capabilities. This allows closing the loop over task planning, as shown in Figure I.7, where now the robot can take high-level semantic goals instead of coordinates of motion goals. We will discuss Spatial AI at length in Part III of this handbook.

## I.5 Handbook Structure

The chapters of this handbook are grouped into three parts.

Part I covers the foundations of SLAM, with particular focus on the estimation-theoretic machinery used in the SLAM back-end and the different types of map representations SLAM can produce. In particular, Chapter 1 introduces the factor-graph formulation of SLAM and reviews how to solve it via iterative nonlinear optimization methods. Then, Chapter 2 takes the indispensable step of extending the formulation to allow the estimation of variables belonging to smooth manifolds, such as rotation and poses. Chapter 3 discusses how to model and mitigate the impact of outliers and incorrect data association in the SLAM back-end. Chapter 4 reviews techniques to make the back-end optimization differentiable, a key step towards interfacing traditional SLAM methods with more recent deep learning architectures. Chapter 5 shifts the focus from the back-end to the question of dense map representations and discusses the most important representations used for SLAM. Finally, Chapter 6 discusses more advanced solvers and theoretical properties of the SLAM back-end.

Part II covers the “state of practice” in SLAM by discussing key approaches and applications of SLAM using different sensing modalities. This part touches on the

SLAM front-end design (which is heavily sensor dependent) and exposes what's feasible with modern SLAM algorithms and systems. Chapter 7 reviews the large body of literature on visual SLAM. Chapter 8 and Chapter 9 cover LiDAR SLAM and radar SLAM, respectively. Chapter 10 discusses recent work on SLAM using event-based cameras. Chapter 11 reviews how to model inertial measurements as part of a factor-graph SLAM system and discusses fundamental limits (*e.g.*, observability). Chapter 12 discussed how to model other sources of odometry information, including wheel and legged odometry.

Part III provides a future-looking view of the state of the art and recent trends. In particular, we touch on a variety of topics, ranging from computational architectures, to novel problems and representations, to the role of language and Foundation Models in SLAM. In particular, Chapter 13 reviews recent improvements obtained by introducing deep learning modules in conjunction with differentiable optimization in SLAM. Chapter 14 discusses opportunities and challenges in using novel map presentations, including neural radiance fields (NeRFs) and Gaussian Splatting. Chapter 15 covers recent work on SLAM in highly dynamic and deformable environments, touching on real applications from mapping in crowded environments to surgical robotics. Chapter 16 discusses progress in Spatial AI and metric-semantic map representations. Chapter 17 considers new opportunities arising from the use of Foundation Models (*e.g.*, Large Vision-Language Models) and their role in creating novel map representation for Spatial AI that allow understanding and grounding “open-vocabulary” commands given in natural language. Finally, Chapter 18 focuses on future computational architectures for Spatial AI that could leverage more flexible and distributed computing hardware and better support spatial perception across many robotic platforms.

# 1

## Factor Graphs for SLAM

Frank Dellaert, Michael Kaess, and Timothy Barfoot

In this chapter we introduce factor graphs and establish the connection with maximum a posteriori (MAP) inference and least squares for the case of Gaussian priors and Gaussian measurement noise. We focus on the SLAM back-end, after measurements have been extracted by the front-end and data association has been accomplished. We discuss both linear and nonlinear optimization methods for the corresponding least squares problems, and then make the connection between sparsity, factor graphs, and Bayes nets more explicit. Finally, we apply this to develop the Bayes tree and the incremental smoothing and mapping (iSAM) algorithm.

*maximum a posteriori*

*iSAM*

### *Historical Note*

A *smoothing* approach to SLAM involves not just the most current robot location, *SLAM* but the entire robot trajectory up to the current time. A number of authors consider the problem of smoothing the robot trajectory only [172, 698, 697, 420, 593, 312], now known as *PoseSLAM*. This is particularly suited to sensors such as laser-range *PoseSLAM* finders that yield pairwise constraints between nearby robot poses.

More generally, one can consider the *full SLAM problem* [1085], i.e., the problem *full SLAM problem* of optimally estimating the entire set of sensor poses along with the parameters of all features in the environment. This led to a flurry of work between 2000 and 2005 where these ideas were applied in the context of SLAM [287, 340, 339, 1085]. From a computational view, this optimization-based smoothing was recognized as beneficial since (a) in contrast to the filtering-based covariance or information matrices, which *both* become fully dense over time [846, 1084], the information matrix associated with smoothing is and stays sparse; (b) in typical mapping scenarios (i.e., not repeatedly traversing a small environment) this matrix is a much more compact representation of the map covariance structure.

*Square-root smoothing and mapping (SAM)*, also known as the ‘factor-graph approach’, was introduced in [249, 254] based on the fact that the information matrix or measurement Jacobian can be efficiently factorized using sparse Cholesky or QR factorization, respectively. This yields a square-root information matrix that can be used to immediately obtain the optimal robot trajectory and map. Factoring the

*Square-root SAM*

information matrix is known in the sequential estimation literature as *square-root information filtering (SRIF)*, and was developed in 1969 for use in JPL’s Mariner 10 missions to Venus [83]. The use of square roots results in more accurate and stable algorithms, and, quoting Maybeck [739], “a number of practitioners have argued, with considerable logic, that square root filters should *always* be adopted in preference to the standard Kalman filter recursion”.

*variable elimination*

Below we discuss in detail how factor graphs are a natural representation for the sparsity inherent in SLAM problems, how (sparse) matrix factorization into a matrix-square root is at the heart of solving these problems, and finally how all this relates to the much more general *variable elimination algorithm*. Much of this chapter is an abridged version of a longer article by Dellaert et al. [255].

## 1.1 Visualizing SLAM With Factor Graphs

In this section we introduce factor graphs as a way of intuitively visualizing the sparse nature of the SLAM problem by first considering a toy example and its factor graph representation. We then show how many different flavors of SLAM can be represented as such, and how even in larger problems the sparse nature of many sparse problems is immediately apparent.

### 1.1.1 A Toy Example

We begin by examining a simple SLAM scenario to illustrate how factor graphs are constructed. Figure 1.1 shows a simple toy example illustrating the structure of the problem graphically. A robot moving across three successive poses  $\mathbf{p}_1$ ,  $\mathbf{p}_2$ , and  $\mathbf{p}_3$  makes bearing observations on two landmarks  $\ell_1$  and  $\ell_2$ . To anchor the solution in space, let us also assume there is an absolute position/orientation measurement on the first pose  $\mathbf{p}_1$ . Without this there would be no information about absolute position, as bearing measurements are all relative.<sup>1</sup>

Because of measurement uncertainty, we cannot hope to recover the true state of the world, but we can obtain a probabilistic description of what can be inferred from the measurements. In the Bayesian probability framework, we use the language of probability theory to assign a subjective degree of belief to uncertain events. We do this using *probability density functions (PDFs)*  $p(\mathbf{x})$  over the unknown variables  $\mathbf{x}$ . PDFs are non-negative functions satisfying

$$\int p(\mathbf{x}) \, d\mathbf{x} = 1, \quad (1.1)$$

which is the axiom of total probability. In the simple example of Figure 1.1, the

<sup>1</sup> Handling rotations properly is a bit more involved than our treatment in this first chapter lets on. However, the next chapter will put us on a proper footing for such quantities. For now, we will assume they are regular vector quantities and delay discussion of their subtleties.

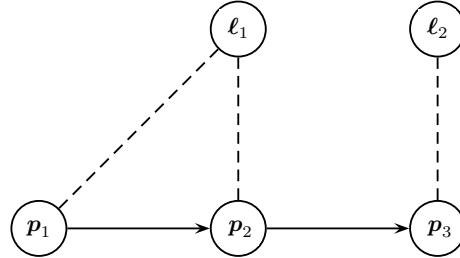


Figure 1.1 A toy simultaneous localization and mapping (SLAM) example with three robot poses and two landmarks. Above we schematically indicate the robot motion with arrows, while the dotted lines indicate bearing measurements.

state,  $\mathbf{x}$ , is

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \ell_1 \\ \ell_2 \end{bmatrix}, \quad (1.2)$$

which is just a stacking of the individual unknowns.

In SLAM we want to characterize our knowledge about the unknowns  $\mathbf{x}$ , in this case robot poses and the unknown landmark positions, when given a set of *observed* measurements  $\mathbf{z}$ . Using the language of Bayesian probability, this is simply the conditional density

$$p(\mathbf{x}|\mathbf{z}), \quad (1.3)$$

and obtaining a description like this is called *probabilistic inference*. A prerequisite is to first specify a probabilistic model for the variables of interest and how they give rise to (uncertain) measurements. This is where *probabilistic graphical models* enter the picture.

Probabilistic graphical models provide a mechanism to compactly describe complex probability densities by exploiting the structure in them [591]. In particular, high-dimensional probability densities can often be factorized as a product of many *factors*, each of which is a probability density over a much smaller domain.

*factors*

### 1.1.2 A Factor-Graph View

Factor graphs are probabilistic graphical models and they allow us to specify a joint density as a product of factors. However, they are more general in that they can be used to specify *any* factored function  $\phi(\mathbf{x})$  over a set of variables  $\mathbf{x}$ , not just probability densities.

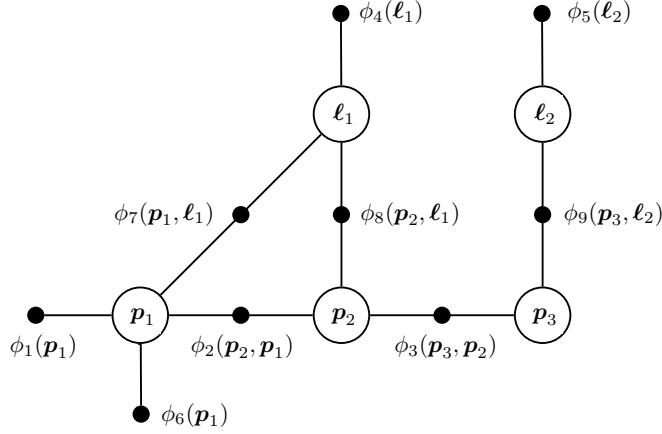


Figure 1.2 Factor graph resulting from the example in Figure 1.1.

To motivate this, consider performing inference for the toy SLAM example. The posterior  $p(\mathbf{x}|\mathbf{z})$  can be re-written using Bayes' law,  $p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x})p(\mathbf{x})$ , as

$$p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{p}_1) p(\mathbf{p}_2|\mathbf{p}_1) p(\mathbf{p}_3|\mathbf{p}_2) \quad (1.4a)$$

$$\times p(\ell_1) p(\ell_2) \quad (1.4b)$$

$$\times p(\mathbf{z}_1|\mathbf{p}_1) \quad (1.4c)$$

$$\times p(\mathbf{z}_2|\mathbf{p}_1, \ell_1) p(\mathbf{z}_3|\mathbf{p}_2, \ell_1) p(\mathbf{z}_4|\mathbf{p}_3, \ell_2). \quad (1.4d)$$

where we assumed a typical Markov chain generative model for the pose trajectory. Each of the factors represents one piece of information about the unknowns,  $\mathbf{x}$ .

*factor graph*

To visualize this factorization, we use a *factor graph*. Figure 1.2 introduces the corresponding factor graph by example: all unknown states  $\mathbf{x}$ , both poses and landmarks, have a node associated with them. Measurements are *not* represented explicitly as they are known, and hence not of interest. In factor graphs we explicitly introduce an additional node type to represent every *factor* in the posterior  $p(\mathbf{x}|\mathbf{z})$ . In the figure, each small black node represents a factor, and—importantly—is connected to only those state variables of which it is a function. For example, the factor  $\phi_9(\mathbf{p}_3, \ell_2)$  is connected only to the variable nodes  $\mathbf{p}_3$  and  $\ell_2$ . In more detail, we have

$$\phi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ell_1, \ell_2) = \phi_1(\mathbf{p}_1) \phi_2(\mathbf{p}_2, \mathbf{p}_1) \phi_3(\mathbf{p}_3, \mathbf{p}_2) \quad (1.5a)$$

$$\times \phi_4(\ell_1) \phi_5(\ell_2) \quad (1.5b)$$

$$\times \phi_6(\mathbf{p}_1) \quad (1.5c)$$

$$\times \phi_7(\mathbf{p}_1, \ell_1) \phi_8(\mathbf{p}_2, \ell_1) \phi_9(\mathbf{p}_3, \ell_2), \quad (1.5d)$$

*factor*

where the correspondence between the factors and the original probability densities in (1.4a)-(1.4d) should be obvious.

The factor values need only be *proportional* to the corresponding probability densities: any normalization constants that do not depend on the state variables may be omitted without consequence. Also, in this example, all factors above came either from a prior, e.g.,  $\phi_1(\mathbf{p}_1) \propto p(\mathbf{p}_1)$  or from a measurement, e.g.,  $\phi_9(\mathbf{p}_3, \ell_2) \propto p(\mathbf{z}_4|\mathbf{p}_3, \ell_2)$ . Although the measurement variables  $\mathbf{z}_1..z_4$  are not explicitly shown in the factor graph, those factors are implicitly conditioned on them. Sometimes, when it helps to make this more explicit, factors can be written as (for example)  $\phi_9(\mathbf{p}_3, \ell_2; \mathbf{z}_4)$  or even  $\phi_{\mathbf{z}_4}(\mathbf{p}_3, \ell_2)$ .

### 1.1.3 Factor Graphs as a Language

In addition to providing a formal basis for inference, factor graphs help visualize SLAM problems of many different flavors, give insight into the structure of the problem, and serve as a *lingua franca* that can help practitioners align across team boundaries. Each factor in a factor graph, such as those in Fig 2.2, can be thought of as an equation involving the variables it is connected to. There are typically many more equations than unknowns, which is why we need to quantify the uncertainty in both prior information and measurements. This will lead to a least squares formulation, appropriately fusing the information from multiple sources.

Many different flavors of the SLAM problem are all easily represented as factor graphs. Figure 1.1 is an example of *landmark-based SLAM* because it involves both pose and landmark variables. Figure 1.3 illustrates several other variants including *bundle adjustment (BA)* (same as landmark-based SLAM but without motion model), *pose-graph optimization (PGO)* (no landmark variables but includes loop closures), and *simultaneous trajectory estimation and mapping (STEAM)* (poses are augmented to include derivatives such as velocity).

The factor graph for a more realistic landmark-based SLAM problem than the toy example could look something like Figure 1.4. This graph was created by simulating a 2D robot, moving in the plane for about 100 time steps, as it observes landmarks. For visualization purposes, each robot pose and landmark is rendered at its ground-truth position in 2D. With this, we see that the odometry factors form a prominent, chain-like backbone, whereas off to the sides binary likelihood factors are connected to the 20 or so landmarks. All factors in such SLAM problems are typically nonlinear, except for priors.

Examining the factor graph reveals a great deal of structure by which we can gain insight into a particular instance of the SLAM problem. First, there are landmarks with a great deal of measurements, which we expect to be pinned down very well. Others have only a tenuous connection to the graph, and hence we expect them to be less well determined. For example, the lone landmark near the bottom right has only a single measurement associated with it: if this is a bearing-only measurement,

*landmark-based SLAM*  
*bundle adjustment (BA)*  
*pose-graph optimization (PGO)*  
*simultaneous trajectory estimation and mapping (STEAM)*

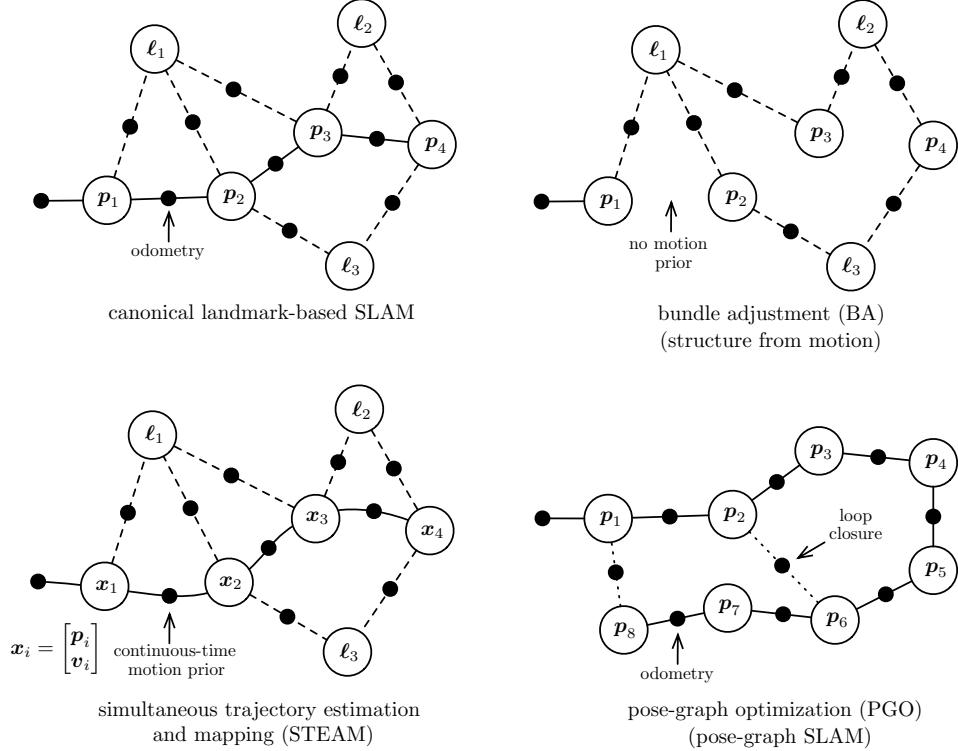


Figure 1.3 A few variants of SLAM problems that can all be viewed through the factor-graph lens. Canonical landmark-based SLAM has both pose and landmark variables; landmarks are measured from poses and there is some motion prior between poses typically based on odometry. BA is the same but without the motion prior. STEAM is similar but now poses can be replaced by higher-order states and a smooth continuous-time motion prior is used. PGO does not have landmark variables but enjoys extra loop closure measurements between poses.

many assignments of a 2D location to the landmark will be equally ‘correct’. This is the same as saying that we have infinite uncertainty in some subset of the domain of the unknowns, which is where prior knowledge should come to the rescue.

## 1.2 From MAP Inference to Least Squares

*maximum a posteriori (MAP) inference*

In SLAM, *maximum a posteriori (MAP) inference* is the process of determining the values for the unknowns  $\mathbf{x}$  that maximally agree with the information present in the uncertain measurements. In real life we are *not* given the ground-truth locations for the landmarks, nor the time-varying pose of the robot, although in many practical cases we might have a good initial estimate. Below we review how to model both prior knowledge and measurements using probability densities, how the posterior

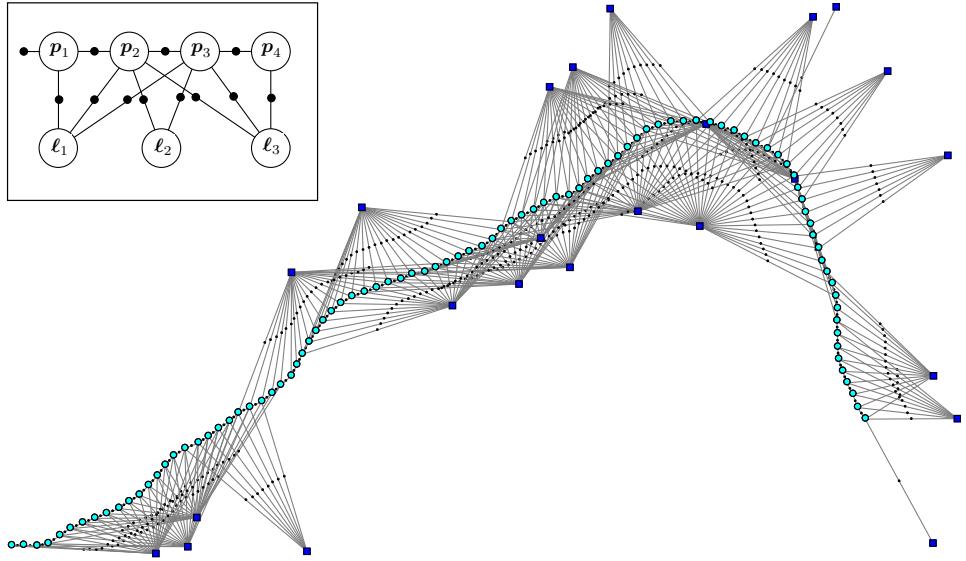


Figure 1.4 Factor graph for a larger, simulated SLAM example.

density given measurements is most conveniently represented as a factor graph, and how given Gaussian priors and Gaussian noise models the corresponding optimization problem is nothing but the familiar nonlinear least squares problem.

### 1.2.1 Factor Graphs for MAP Inference

We are interested in the *unknown state variables*  $\mathbf{x}$ , such as poses and/or landmarks, *given* the measurements  $\mathbf{z}$ . The most-often-used *estimator* for these unknown state variables  $\mathbf{x}$  is the *maximum a posteriori (MAP) estimate*, so named because it maximizes the posterior density  $p(\mathbf{x}|\mathbf{z})$  of the states  $\mathbf{x}$  given the measurements  $\mathbf{z}$ :

$$\mathbf{x}^{\text{MAP}} = \arg \max_{\mathbf{x}} p(\mathbf{x}|\mathbf{z}) \quad (1.6a)$$

$$= \arg \max_{\mathbf{x}} \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z})} \quad (1.6b)$$

$$= \arg \max_{\mathbf{x}} p(\mathbf{z}|\mathbf{x})p(\mathbf{x}) \quad (1.6c)$$

The second equation above is Bayes' law, and expresses the posterior as the product of the measurement density  $p(\mathbf{z}|\mathbf{x})$  and the prior  $p(\mathbf{x})$  over the states, appropriately normalized by the factor  $p(\mathbf{z})$ . The third equation drops the  $p(\mathbf{z})$  since this does not depend on the  $\mathbf{x}$  and therefore will not impact the arg max operation.

We use factor graphs to express the unnormalized posterior  $p(\mathbf{z}|\mathbf{x})p(\mathbf{x})$ . Formally a factor graph is a bipartite graph  $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$  with two types of nodes: *factors*

$\phi_i \in \mathcal{U}$  and variables  $\mathbf{x}_j \in \mathcal{V}$ . Edges  $e_{ij} \in \mathcal{E}$  are always between factor nodes and variables nodes. The set of variable nodes adjacent to a factor  $\phi_i$  is written as  $\mathcal{X}(\phi_i)$ , and we write  $\mathbf{x}_i$  for an assignment to this set. With these definitions, a factor graph  $F$  defines the factorization of a global function  $\phi(\mathbf{x})$  as

$$\phi(\mathbf{x}) = \prod_i \phi_i(\mathbf{x}_i). \quad (1.7)$$

In other words, the independence relationships are encoded by the edges  $e_{ij}$  of the factor graph, with each factor  $\phi_i$  a function of *only* the variables  $\mathbf{x}_i$  in its adjacency set  $\mathcal{X}(\phi_i)$ .

In the rest of this chapter, we show how to find an optimal assignment, the MAP estimate, through optimization over the unknown variables in the factor graph. Indeed, for an arbitrary factor graph, MAP inference comes down to maximizing the product (1.7) of all factor-graph potentials:

$$\mathbf{x}^{\text{MAP}} = \arg \max_{\mathbf{x}} \phi(\mathbf{x}) \quad (1.8a)$$

$$= \arg \max_{\mathbf{x}} \prod_i \phi_i(\mathbf{x}_i). \quad (1.8b)$$

What is left now is to derive the exact form of the factors  $\phi_i(\mathbf{x}_i)$ , which depends very much on how we model the measurement models  $p(\mathbf{z}|\mathbf{x})$  and the prior densities  $p(\mathbf{x})$ . We discuss this in detail next.

### 1.2.2 Specifying Probability Densities

The exact form of the densities  $p(\mathbf{z}|\mathbf{x})$  and  $p(\mathbf{x})$  above depends very much on the application and the sensors used. The most often used densities involve the *multivariate Gaussian density*, with probability density

$$\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\Sigma}}^2\right), \quad (1.9)$$

where  $\boldsymbol{\mu} \in \mathbb{R}^n$  is the mean,  $\boldsymbol{\Sigma}$  is an  $n \times n$  covariance matrix, and

$$\|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\Sigma}}^2 \triangleq (\boldsymbol{\theta} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}) \quad (1.10)$$

denotes the squared Mahalanobis distance. The normalization constant  $\sqrt{|2\pi\boldsymbol{\Sigma}|} = (2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}$ , where  $|.|$  denotes the matrix determinant, ensures the multivariate Gaussian density integrates to 1.0 over its domain.

Priors on unknown quantities are often specified using a Gaussian density, and in many cases it is both justified and convenient to model measurements as corrupted by zero-mean Gaussian noise. For example, a bearing measurement<sup>2</sup> from a given

<sup>2</sup> As a reminder, there are some subtleties associated with rotational state variables that we will discuss more thoroughly in the next chapter.

pose  $\mathbf{p}$  to a given landmark  $\ell$  would be modeled as

$$\mathbf{z} = \mathbf{h}(\mathbf{p}, \ell) + \boldsymbol{\eta}, \quad (1.11)$$

where  $\mathbf{h}(\cdot)$  is a *measurement prediction function*, and the noise  $\boldsymbol{\eta}$  is drawn from a zero-mean Gaussian density with measurement covariance  $\Sigma_R$ . This yields the following conditional density  $p(\mathbf{z}|\mathbf{p}, \ell)$  on the measurement  $\mathbf{z}$ :

$$p(\mathbf{z}|\mathbf{p}, \ell) = \mathcal{N}(\mathbf{z}; \mathbf{h}(\mathbf{p}, \ell), \Sigma_R) = \frac{1}{\sqrt{|2\pi\Sigma_R|}} \exp\left(-\frac{1}{2}\|\mathbf{z} - \mathbf{h}(\mathbf{p}, \ell)\|_{\Sigma_R}^2\right). \quad (1.12)$$

The measurement functions  $\mathbf{h}(\cdot)$  are often nonlinear in practical robotics applications. Still, while they depend on the sensor used and the SLAM front-end, they are typically not difficult to reason about or write down. The measurement function for a 2D bearing measurement is simply

$$\mathbf{h}(\mathbf{p}, \ell) = \text{atan2}(\ell_y - p_y, \ell_x - p_x), \quad (1.13)$$

where  $\text{atan2}$  is the well-known two-argument arctangent variant. Hence, the final probabilistic *measurement model*  $p(\mathbf{z}|\mathbf{p}, \ell)$  is obtained as

$$p(\mathbf{z}|\mathbf{p}, \ell) = \frac{1}{\sqrt{|2\pi\Sigma_R|}} \exp\left(-\frac{1}{2}\|\mathbf{z} - \text{atan2}(\ell_y - p_y, \ell_x - p_x)\|_{\Sigma_R}^2\right). \quad (1.14)$$

Note that we will not *always* assume Gaussian measurement noise: to cope with the occasional data association mistake, for example, many authors have proposed the use of robust measurement densities, with heavier tails than a Gaussian density; these are discussed in Chapter 3.

Not all probability densities involved are derived from measurements. For example, in the toy SLAM problem the prior  $p(\mathbf{x})$  on the trajectory is made up of a prior  $p(\mathbf{p}_1)$  and conditional densities  $p(\mathbf{p}_{t+1}|\mathbf{p}_t)$ , specifying a probabilistic *motion model*  $p(\mathbf{p}_{t+1}|\mathbf{p}_t, \mathbf{u}_t)$  that the robot is assumed to obey given known control inputs  $\mathbf{u}_t$ . In practice, we often use a conditional Gaussian assumption,

$$p(\mathbf{p}_{t+1}|\mathbf{p}_t, \mathbf{u}_t) = \frac{1}{\sqrt{|2\pi\Sigma_Q|}} \exp\left(-\frac{1}{2}\|\mathbf{p}_{t+1} - \mathbf{g}(\mathbf{p}_t, \mathbf{u}_t)\|_{\Sigma_Q}^2\right), \quad (1.15)$$

where  $\mathbf{g}(\cdot)$  is a motion model, and  $\Sigma_Q$  a covariance matrix of the appropriate dimensionality, e.g.,  $3 \times 3$  in the case of robots operating in the plane.

Often we have no known control inputs  $\mathbf{u}_t$  but instead we *measure* how the robot moved, e.g., via an odometry measurement  $\mathbf{o}_t$ . For example, if we assume the odometry simply measures the difference between poses, subject to Gaussian noise with covariance  $\Sigma_S$ , we obtain

$$p(\mathbf{o}_t|\mathbf{p}_{t+1}, \mathbf{p}_t) = \frac{1}{\sqrt{|2\pi\Sigma_S|}} \exp\left(-\frac{1}{2}\|\mathbf{o}_t - (\mathbf{p}_{t+1} - \mathbf{p}_t)\|_{\Sigma_S}^2\right). \quad (1.16)$$

If we have *both* known control inputs  $\mathbf{u}_t$  and odometry measurements  $\mathbf{o}_t$  we can combine (1.15) and (1.16).

Note that for robots operating in three-dimensional space, we will need slightly more sophisticated machinery to specify densities on nonlinear manifolds such as SE(3), as discussed in the next chapter.

### 1.2.3 Nonlinear Least Squares

We now show that MAP inference for SLAM problems with Gaussian noise models as above is equivalent to solving a nonlinear least squares problem. If we assume that all factors are of the form

$$\phi_i(\mathbf{x}_i) \propto \exp\left(-\frac{1}{2} \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2\right), \quad (1.17)$$

which include both simple Gaussian priors and likelihood factors derived from measurements corrupted by zero-mean, normally distributed noise. Taking the negative log of (1.8b) and dropping the factor  $\frac{1}{2}$  allows us to instead minimize a sum of *nonlinear least squares* terms:

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2. \quad (1.18)$$

Minimizing this objective function performs sensor fusion through the process of combining several measurement-derived factors, and possibly several priors, to uniquely determine the MAP solution for the unknowns.

An important and non-obvious observation is that the factors in (1.18) typically represent rather *under-specified* densities on the involved unknown variables  $\mathbf{x}_i$ . Indeed, except for simple prior factors, the measurements  $\mathbf{z}_i$  are typically of lower dimension than the unknowns  $\mathbf{x}_i$ . In those cases, the factor by itself accords the same likelihood to an infinite subset of the domain of  $\mathbf{x}_i$ . For example, a 2D measurement in a camera image is consistent with an entire ray of 3D points that project to the same image location.

Even though the functions  $\mathbf{h}_i$  are nonlinear, *if* we have a decent initial guess available, then the *nonlinear optimization* methods we discuss in this chapter will be able to converge to the global minimum of (1.18). We should caution, however, that as our objective in (1.18) is *non-convex*, there is no guarantee that we will not get stuck in a local minimum if our initial guess is poor. This has led to so-called *certifiably optimal solvers*, which are the subject of a later chapter. Below, however, we focus on local methods rather than global solvers. We start off below by considering the easier problem of solving a *linearized* version of the problem.

*certifiably optimal solvers*  
*linearized*

## 1.3 Solving Linear Least Squares

Before tackling the more difficult problem of nonlinear least squares, in this section we first show how to *linearize* the problem, show how this leads to a *linear least squares*

*linear least squares*

problem, and review matrix factorization as computationally efficient way to solve the corresponding *normal equations*. A seminal reference for these methods is the *normal equations* book by Golub and Loan [389].

### 1.3.1 Linearization

We can linearize all measurement functions  $\mathbf{h}_i(\cdot)$  in the nonlinear least squares objective function (1.18) using a simple Taylor expansion,

$$\mathbf{h}_i(\mathbf{x}_i) = \mathbf{h}_i(\mathbf{x}_i^0 + \boldsymbol{\delta}_i) \approx \mathbf{h}_i(\mathbf{x}_i^0) + \mathbf{H}_i \boldsymbol{\delta}_i, \quad (1.19)$$

where the *measurement Jacobian*  $\mathbf{H}_i$  is defined as the (multivariate) partial derivative of  $\mathbf{h}_i(\cdot)$  at a given linearization point  $\mathbf{x}_i^0$ , *measurement Jacobian*

$$\mathbf{H}_i \triangleq \left. \frac{\partial \mathbf{h}_i(\mathbf{x}_i)}{\partial \mathbf{x}_i} \right|_{\mathbf{x}_i^0}, \quad (1.20)$$

and  $\boldsymbol{\delta}_i \triangleq \mathbf{x}_i - \mathbf{x}_i^0$  is the *state update vector*. Note that we make an assumption that  $\mathbf{x}_i$  lives in a vector space or, equivalently, can be represented by a *vector*. This is not always the case, e.g., when some of the unknown states in  $\mathbf{x}$  represent 3D rotations or other more complex manifold types. We will revisit this issue in Chapter 2.

Substituting the Taylor expansion (1.19) into the nonlinear least squares expression (1.18) we obtain a *linear least squares* problem in the state update vector  $\boldsymbol{\delta}$ , *linear least squares*

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0) - \mathbf{H}_i \boldsymbol{\delta}_i\|_{\Sigma_i}^2 \quad (1.21a)$$

$$= \arg \min_{\boldsymbol{\delta}} \sum_i \|(\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)) - \mathbf{H}_i \boldsymbol{\delta}_i\|_{\Sigma_i}^2, \quad (1.21b)$$

where  $\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)$  is the *prediction error* at the linearization point, *i.e.*, the difference between actual and predicted measurement. Above,  $\boldsymbol{\delta}^*$  denotes the solution to the locally linearized problem.

By a simple change of variables we can drop the covariance matrices  $\Sigma_i$  from this point forward: defining  $\Sigma^{1/2}$  as the matrix square root of  $\Sigma$ , we can rewrite the square Mahalanobis norm as follows:

$$\|e\|_{\Sigma}^2 \triangleq e^\top \Sigma^{-1} e = \left( \Sigma^{-1/2} e \right)^\top \left( \Sigma^{-1/2} e \right) = \left\| \Sigma^{-1/2} e \right\|_2^2. \quad (1.22)$$

Hence, we can eliminate the covariances  $\Sigma_i$  by pre-multiplying the Jacobian  $\mathbf{H}_i$  and the prediction error in each term in (1.21b) with  $\Sigma_i^{-1/2}$ :

$$\mathbf{A}_i = \Sigma_i^{-1/2} \mathbf{H}_i \quad (1.23a)$$

$$\mathbf{b}_i = \Sigma_i^{-1/2} (\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)). \quad (1.23b)$$

This process is a form of *whitening*. For example, in the case of scalar measurements

it simply means dividing each term by the measurement standard deviation  $\sigma_i$ . Note that this eliminates the units of the measurements (e.g., length, angles) so that the different rows can be combined into a single cost function.

### 1.3.2 SLAM as Least Squares

After linearization, we finally obtain the following standard least squares problem:

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{A}_i \boldsymbol{\delta}_i - \mathbf{b}_i\|_2^2 \quad (1.24a)$$

$$= \arg \min_{\boldsymbol{\delta}} \|\mathbf{A} \boldsymbol{\delta} - \mathbf{b}\|_2^2, \quad (1.24b)$$

Above  $\mathbf{A}$  and  $\mathbf{b}$  are obtained by collecting all whitened Jacobian matrices  $\mathbf{A}_i$  and whitened prediction errors  $\mathbf{b}_i$  into one large matrix  $\mathbf{A}$  and right-hand-side (RHS) vector  $\mathbf{b}$ , respectively.

The Jacobian  $\mathbf{A}$  is a large-but-sparse matrix, with a block structure that mirrors the structure of the underlying factor graph. We will examine this sparsity structure in detail below. First, however, we review the classical linear algebra approach to solving this least squares problem.

### 1.3.3 Matrix Factorization for Least Squares

For a full-rank  $m \times n$  matrix  $\mathbf{A}$ , with  $m \geq n$ , the unique least squares solution to (1.24b) can be found by solving the *normal equations*:

$$(\mathbf{A}^\top \mathbf{A}) \boldsymbol{\delta}^* = \mathbf{A}^\top \mathbf{b}. \quad (1.25)$$

*information matrix*  
*Hessian*

*Cholesky factor*  
*Cholesky factorization*

This is normally done by factoring the *information matrix*  $\mathbf{\Lambda}$ , also called the *Hessian* matrix, defined and factored as follows:

$$\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A} = \mathbf{R}^\top \mathbf{R}. \quad (1.26)$$

Above, the *Cholesky factor*  $\mathbf{R}$  is an upper-triangular  $n \times n$  matrix<sup>3</sup> and is computed using *Cholesky factorization*, a variant of lower-upper (LU) factorization for symmetric positive-definite matrices. After this,  $\boldsymbol{\delta}^*$  can be found by solving first

$$\mathbf{R}^\top \mathbf{y} = \mathbf{A}^\top \mathbf{b} \quad (1.27)$$

for  $\mathbf{y}$  and then

$$\mathbf{R} \boldsymbol{\delta}^* = \mathbf{y} \quad (1.28)$$

for  $\boldsymbol{\delta}^*$  by forward and backward substitution, respectively. For dense matrices, Cholesky factorization requires  $n^3/3$  flops, and the entire algorithm, including computing half of the symmetric  $\mathbf{A}^\top \mathbf{A}$ , requires  $(m + n/3)n^2$  flops. One could also use

<sup>3</sup> Some treatments, including [389], define the Cholesky triangle as the lower-triangular matrix  $\mathbf{L} = \mathbf{R}^\top$ , but the other convention is more convenient here.

lower-diagonal-upper (LDU) factorization, a variant of Cholesky decomposition that avoids the computation of square roots.

An alternative to Cholesky factorization that is more accurate and more numerically stable is to proceed via *QR-factorization*, which works *without* computing the information matrix  $\mathbf{\Lambda}$ . Instead, we compute the QR-factorization of  $\mathbf{A}$  itself along with its corresponding RHS:

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix} = \mathbf{Q}^\top \mathbf{b}. \quad (1.29)$$

Here  $\mathbf{Q}$  is an  $m \times m$  orthogonal matrix,  $\mathbf{d} \in \mathbb{R}^n$ ,  $\mathbf{e} \in \mathbb{R}^{m-n}$ , and  $\mathbf{R}$  is the *same* upper-triangular Cholesky triangle. The preferred method for factorizing a dense matrix  $\mathbf{A}$  is to compute  $\mathbf{R}$  column by column, proceeding from left to right. For each column  $j$ , all nonzero elements below the diagonal are zeroed out by multiplying  $\mathbf{A}$  on the left with a *Householder reflection matrix*  $\mathbf{H}_j$ . After  $n$  iterations  $\mathbf{A}$  is completely factorized:

$$\mathbf{H}_n \cdots \mathbf{H}_2 \mathbf{H}_1 \mathbf{A} = \mathbf{Q}^\top \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}. \quad (1.30)$$

The orthogonal matrix  $\mathbf{Q}$  is not usually formed: instead, the transformed RHS  $\mathbf{Q}^\top \mathbf{b}$  is computed by appending  $\mathbf{b}$  as an extra column to  $\mathbf{A}$ . Because the  $\mathbf{Q}$  factor is orthogonal, we have

$$\|\mathbf{A} \boldsymbol{\delta} - \mathbf{b}\|_2^2 = \|\mathbf{Q}^\top \mathbf{A} \boldsymbol{\delta} - \mathbf{Q}^\top \mathbf{b}\|_2^2 = \|\mathbf{R} \boldsymbol{\delta} - \mathbf{d}\|_2^2 + \|\mathbf{e}\|_2^2, \quad (1.31)$$

where we made use of the equalities from (1.29). Clearly,  $\|\mathbf{e}\|_2^2$  will be the least squares sum of squared residuals, and the least squares solution  $\boldsymbol{\delta}^*$  can be obtained by solving the triangular system

$$\mathbf{R} \boldsymbol{\delta}^* = \mathbf{d} \quad (1.32)$$

via back-substitution. Note that the upper-triangular factor  $\mathbf{R}$  obtained using QR factorization is the same (up to possible sign changes on the diagonal) as would be obtained by Cholesky factorization, since

$$\mathbf{A}^\top \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}^\top \mathbf{Q}^\top \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} = \mathbf{R}^\top \mathbf{R}, \quad (1.33)$$

where we again made use of the fact that  $\mathbf{Q}$  is orthogonal. The cost of QR is dominated by the cost of the Householder reflections, which is  $2(m-n/3)n^2$ . Comparing this with Cholesky, we see that both algorithms require  $O(mn^2)$  operations when  $m \gg n$ , but that QR-factorization is slower by a factor of 2.

In summary, the linearized optimization problem associated with SLAM can be concisely stated in terms of basic linear algebra. It comes down to factorizing either the information matrix  $\mathbf{\Lambda}$  or the measurement Jacobian  $\mathbf{A}$  into square-root form. Because they are based on matrix square roots derived from the *SAM* problem, we

have referred to this family of approaches as *square-root SAM*, or  $\sqrt{\text{SAM}}$  for short [249, 254].

## 1.4 Nonlinear Optimization

In this section, we discuss some classic optimization approaches to the nonlinear least squares problem defined in (1.18). As a reminder, in SLAM the nonlinear least squares objective function is given by

$$J(\mathbf{x}) \triangleq \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2 \quad (1.34)$$

and corresponds to a nonlinear factor graph derived from the measurements along with prior densities on some or all unknowns.

Nonlinear least squares problems cannot be solved directly in general, but require an iterative solution starting from a suitable initial estimate. Nonlinear optimization methods do so by solving a succession of linear approximations to (1.18) in order to approach the minimum [264]. A variety of algorithms exist that differ in how they locally approximate the cost function, and in how they find an improved estimate based on that local approximation. A general in-depth treatment of nonlinear solvers is provided by [811], while [389] focuses on the linear-algebra perspective.

All of the algorithms share the following basic structure: they start from an initial estimate  $\mathbf{x}^0$ . In each iteration, an update step  $\boldsymbol{\delta}$  is calculated and applied to obtain the next estimate  $\mathbf{x}^{t+1} = \mathbf{x}^t + \boldsymbol{\delta}$ . This process ends when certain convergence criteria are reached, such as the norm of the change  $\boldsymbol{\delta}$  falling below a small threshold.

### 1.4.1 Steepest Descent

Steepest Descent (SD) uses the direction of steepest descent at the current estimate to calculate the following update step:

$$\boldsymbol{\delta}^{\text{sd}} = -\alpha \nabla J(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^t}. \quad (1.35)$$

Here the negative gradient is used to identify the direction of steepest descent. For the nonlinear least squares objective function (1.34), we locally approximate the objective function as a quadratic,  $J(\mathbf{x}) \approx \|\mathbf{A}(\mathbf{x} - \mathbf{x}^t) - \mathbf{b}\|_2^2$ , and obtain the exact gradient  $\nabla J(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^t} = -2\mathbf{A}^\top \mathbf{b}$  at the linearization point  $\mathbf{x}^t$ .

The step size  $\alpha$  needs to be carefully chosen to balance between safe updates and reasonable convergence speed. An explicit line search can be performed to find a minimum in the given direction. SD is a simple algorithm, but suffers from slow convergence near the minimum.

### 1.4.2 Gauss-Newton

Gauss-Newton (GN) provides faster convergence by using a second-order update. GN exploits the special structure of the nonlinear least squares problem to approximate the Hessian using the Jacobian as  $\mathbf{A}^\top \mathbf{A}$ . The GN update step is obtained by solving the normal equations (1.25),

$$\mathbf{A}^\top \mathbf{A} \boldsymbol{\delta}^{\text{gn}} = \mathbf{A}^\top \mathbf{b}, \quad (1.36)$$

by any of the methods in Section 1.3.3. For a well-behaved (i.e., nearly quadratic) objective function and a good initial estimate, Gauss-Newton exhibits nearly quadratic convergence. If the quadratic fit is poor, a GN step can lead to a new estimate that is further from the minimum and subsequent divergence.

### 1.4.3 Levenberg-Marquardt

The Levenberg-Marquardt (LM) algorithm allows for iterating multiple times to convergence while controlling in which region one is willing to trust the quadratic approximation made by Gauss-Newton. Hence, such a method is often called a *trust-region method*.

To combine the advantages of both the SD and GN methods, Levenberg [644] proposed to modify the normal equations (1.25) by adding a non-negative constant  $\lambda \in \mathbb{R}^+ \cup \{0\}$  to the diagonal

$$(\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I}) \boldsymbol{\delta}^{\text{lm}} = \mathbf{A}^\top \mathbf{b}. \quad (1.37)$$

Note that for  $\lambda = 0$  we obtain GN, and for large  $\lambda$  we approximately obtain  $\boldsymbol{\delta}^* \approx \frac{1}{\lambda} \mathbf{A}^\top \mathbf{b}$ , an update in the negative gradient direction of the cost function  $J$  (1.34). Hence, LM can be seen to blend naturally between the GN and SD methods.

Marquardt [728] later proposed to take into account the scaling of the diagonal entries to provide faster convergence:

$$(\mathbf{A}^\top \mathbf{A} + \lambda \text{diag}(\mathbf{A}^\top \mathbf{A})) \boldsymbol{\delta}^{\text{lm}} = \mathbf{A}^\top \mathbf{b}. \quad (1.38)$$

This modification causes larger steps in the steepest-descent direction if the gradient is small (nearly flat directions of the objective function), because there the inverse of the diagonal entries will be large. Conversely, in steep directions of the objective function the algorithm becomes more cautious and takes smaller steps. Both modifications of the normal equations can be interpreted in Bayesian terms as adding a zero-mean prior to the system.

A key difference between GN and LM is that the latter rejects updates that would lead to a higher sum of squared residuals. A rejected update means that the nonlinear function is locally not well-behaved, and smaller steps are needed. This is achieved by heuristically increasing the value of  $\lambda$ , for example by multiplying its current value by a factor of 10, and resolving the modified normal equations.

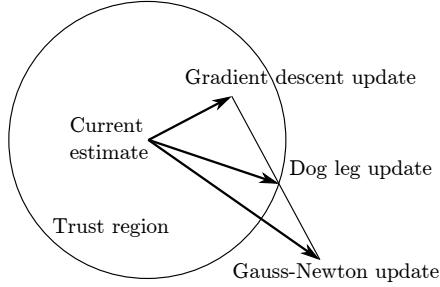


Figure 1.5 Powell’s dogleg algorithm combines the separately computed Gauss-Newton and gradient descent update steps.

On the other hand, if a step leads to a reduction of the sum of squared residuals, it is accepted, and the state estimate is updated accordingly. In this case,  $\lambda$  is reduced (*e.g.*, by dividing by a factor of 10), and the algorithm repeats with a new linearization point, until convergence.

#### 1.4.4 Dogleg Minimization

Powell’s dogleg (PDL) algorithm [882] can be a more efficient alternative to LM [694]. A major disadvantage of the Levenberg-Marquardt algorithm is that in case a step gets rejected, the modified information matrix has to be refactored, which is the most expensive component of the algorithm. Hence, the key idea behind PDL is to separately compute the GN and SD steps, and then combine appropriately. If the LM step gets rejected, the directions of the GN and SD steps are still valid, and they can be combined in a different way until a reduction in the cost is achieved. Hence, each update of the state estimate only involves one matrix factorization, as opposed to several.

Figure 1.5 shows how the GN and SD steps are combined. The combined step starts with the SD update, followed by a sharp bend (hence the term dogleg) towards the GN update, but stopping at the trust region boundary. Unlike LM, PDL maintains an explicit trust region within which we trust the linear assumption. The appropriateness of the linear approximation is determined by the gain ratio

$$\rho = \frac{J(\mathbf{x}^t) - J(\mathbf{x}^t + \boldsymbol{\delta})}{L(\mathbf{0}) - L(\boldsymbol{\delta})}, \quad (1.39)$$

where  $L(\boldsymbol{\delta}) = \mathbf{A}^\top \mathbf{A} \boldsymbol{\delta} - \mathbf{A}^\top \mathbf{b}$  is the linearization of the nonlinear quadratic cost function  $J$  from (1.34) at the current estimate  $\mathbf{x}^t$ . If  $\rho$  is small (*i.e.*,  $\rho < 0.25$ ) then the cost has not reduced as predicted by the linearization and the trust region is reduced. On the other hand, if the reduction is as predicted (or better, *i.e.*,  $\rho > 0.75$ ), then the trust region is increased depending on the magnitude of the update vector, and the step is accepted.

## 1.5 Factor Graphs and Sparsity

The solvers presented so far assume that the matrices involved may be dense. Dense methods will not scale to realistic problem sizes in SLAM. For the toy problem in Figure 1.1 a dense method will work fine. The larger simulation example, with its factor graph shown in Figure 1.4, is more representative of real-world problems. However, it is still relatively small as real SLAM problems go, where problems with thousands or even millions of unknowns are common. Yet, we are able to handle these without a problem because of sparsity.

The sparsity can be appreciated directly from looking at the factor graph. It is clear from Figure 1.4 that the graph is *sparse* (i.e., it is by no means a fully connected graph). The odometry chain linking the 100 unknown poses is a linear structure of 100 binary factors, instead of the possible  $100^2$  (binary) factors. In addition, with 20 landmarks we could have up to 2000 factors linking each landmark to each pose: the true number is closer to 400. And finally, there are no factors between landmarks at all. This reflects that we have not been given any information about their relative position. This structure is typical of most SLAM problems.

### 1.5.1 The Sparse Jacobian and its Factor Graph

The key to modern SLAM algorithms is exploiting sparsity, and an important property of factor graphs in SLAM is that they represent the sparse block structure in the resulting sparse Jacobian matrix  $\mathbf{A}$ . To see this, let us revisit the least squares problem that is the key computation in the inner loop of the nonlinear SLAM problem:

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{A}_i \boldsymbol{\delta}_i - \mathbf{b}_i\|_2^2. \quad (1.40)$$

Each term above is derived from a factor in the original, nonlinear SLAM problem, linearized around the current linearization point (1.21b). The matrices  $\mathbf{A}_i$  can be broken up in blocks corresponding to each variable, and collected in a large, block-sparse Jacobian whose sparsity structure is given exactly by the factor graph.

Even though these linear problems typically arise as inner iterations in nonlinear optimization, we drop the  $\boldsymbol{\delta}$  notation below, as everything holds for general linear problems regardless of their origin.

Consider the factor graph for the small toy example in Figure 1.1. After linearization, we obtain a sparse system  $[\mathbf{A}|\mathbf{b}]$  with the block structure in Figure 1.6. Comparing this with the factor graph, it is obvious that every factor corresponds to a block-row, and every variable corresponds to a block-column of  $\mathbf{A}$ . In total there are nine block-rows, one for every factor in the factorization of  $\phi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ell_1, \ell_2)$ .

$$[\mathbf{A}|\mathbf{b}] = \begin{array}{c|ccccc|c} & \delta\ell_1 & \delta\ell_2 & \delta\mathbf{p}_1 & \delta\mathbf{p}_2 & \delta\mathbf{p}_3 & \mathbf{b} \\ \phi_1 & & & \mathbf{A}_{13} & & & \mathbf{b}_1 \\ \phi_2 & & & \mathbf{A}_{23} & \mathbf{A}_{24} & & \mathbf{b}_2 \\ \phi_3 & & & & \mathbf{A}_{34} & \mathbf{A}_{35} & \mathbf{b}_3 \\ \phi_4 & \mathbf{A}_{41} & & & & & \mathbf{b}_4 \\ \phi_5 & & \mathbf{A}_{52} & & & & \mathbf{b}_5 \\ \phi_6 & & & \mathbf{A}_{63} & & & \mathbf{b}_6 \\ \phi_7 & & & \mathbf{A}_{73} & & & \mathbf{b}_7 \\ \phi_8 & & & & \mathbf{A}_{84} & & \mathbf{b}_8 \\ \phi_9 & & & \mathbf{A}_{92} & & \mathbf{A}_{95} & \mathbf{b}_9 \end{array}$$

Figure 1.6 Block structure of the sparse Jacobian  $\mathbf{A}$  for the toy SLAM example in Figure 1.1 with  $\boldsymbol{\delta} = [\delta\ell_1^\top \delta\ell_2^\top \delta\mathbf{p}_1^\top \delta\mathbf{p}_2^\top \delta\mathbf{p}_3^\top]^\top$ . Blank entries are zeros.

$$\left[ \begin{array}{ccc} \mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{13} & \mathbf{\Lambda}_{14} \\ & \mathbf{\Lambda}_{22} & & \mathbf{\Lambda}_{25} \\ \mathbf{\Lambda}_{31} & \mathbf{\Lambda}_{33} & \mathbf{\Lambda}_{34} \\ \mathbf{\Lambda}_{41} & \mathbf{\Lambda}_{43} & \mathbf{\Lambda}_{44} & \mathbf{\Lambda}_{45} \\ & \mathbf{\Lambda}_{52} & \mathbf{\Lambda}_{54} & \mathbf{\Lambda}_{55} \end{array} \right]$$

Figure 1.7 The information matrix  $\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A}$  for the toy SLAM problem.

### 1.5.2 The Sparse Information Matrix and its Graph

When using Cholesky factorization for solving the normal equations, as explained in Section 1.3.3, we first form the Hessian or information matrix  $\mathbf{\Lambda} = \mathbf{A}^\top \mathbf{A}$ .<sup>4</sup> In general, since the Jacobian  $\mathbf{A}$  is block-sparse, the Hessian  $\mathbf{\Lambda}$  is expected to be sparse as well. By construction, the Hessian is a symmetric matrix, and if a unique MAP solution to the problem exists, it is also positive definite.

*undirected graphical model* *Markov random field* *Markov random field*

The information matrix  $\mathbf{\Lambda}$  can be associated with another, *undirected graphical model* for the SLAM problem, namely a *Markov random field* or *Markov random field (MRF)*. In contrast to a factor graph, an MRF is a graphical model that involves only the variables. The graph  $G$  of an MRF is an undirected graph: the edges only indicate that there is *some* interaction between the variables involved. At the block level, the sparsity pattern of  $\mathbf{\Lambda} = \mathbf{A}^\top \mathbf{A}$  is exactly the adjacency matrix of  $G$ .

Figure 1.7 shows the information matrix  $\mathbf{\Lambda}$  associated with our running toy example. In this case, there are five variables that partition the Hessian as shown. The zero blocks indicate which variables do not interact (e.g.,  $\ell_1$  and  $\ell_2$  have no direct interaction). Figure 1.8 shows the corresponding MRF.

In what follows, we will frequently refer to the undirected graph  $G$  of the MRF

<sup>4</sup> Note that  $\mathbf{A}^\top \mathbf{A}$  is not true Hessian, but is often used to approximate Hessian by truncating a Taylor series of the residual.

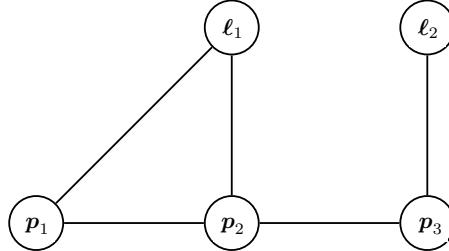


Figure 1.8 The Hessian matrix  $\Lambda$  can be interpreted as the matrix associated with the Markov random field representation for the problem.

associated with an inference problem. However, we will not use the MRF graphical model much beyond that as factor graphs are better suited to our needs. They are able to express a finer-grained factorization, and are more closely related to the original problem formulation. For example, if there exist ternary (or higher) factors in the factor graph, the graph  $G$  of the equivalent MRF connects those nodes in an undirected clique (a group of fully connected variables), but the origin of the corresponding clique potential is lost. In linear algebra, this reflects the fact that many matrices  $\mathbf{A}$  can yield the same  $\Lambda = \mathbf{A}^\top \mathbf{A}$  matrix: important information on the sparsity is lost.

### 1.5.3 Sparse Factorization

We have seen MAP estimation amounts to solving a linear system of equations as described in Section 1.3.3. In the case of nonlinear least squares problems, we solve such a system repeatedly in an iterative setup. We have seen in the previous two sections that both  $\mathbf{A}$  and  $\mathbf{A}^\top \mathbf{A}$  enjoy sparsity determined by the factor graph and MRF connectivity, respectively. Without going into detail, this known sparsity pattern can be used to greatly speed up either Cholesky factorization (in the case of working with  $\mathbf{A}^\top \mathbf{A}$ ) or QR-factorization (in the case of working with  $\mathbf{A}$ ). Efficient software implementations are available, e.g., CHOLMOD [189] and SuiteSparseQR [240], which are also used under the hood by several software packages. In practice, sparse Cholesky or LDU factorization outperform QR factorization on sparse problems as well, and not just by a constant factor.

The flop count for sparse factorization will be much lower than for a dense matrix. Crucially, the column ordering chosen for the sparse matrices can dramatically influence the total flop-count. While any order will ultimately produce an identical MAP estimate, the variable order determines the *fill-in* of matrix factors (i.e., the extra nonzero entries beyond the sparsity pattern of the matrix being factored). It is known that finding the variable ordering that minimizes fill-in during matrix fac-

torization is an NP-hard problem [1236], so we must resort to using good heuristics. This will in turn affect the computational complexity of the factorization algorithm.

We demonstrate this by way of an example. Recall the larger simulation example, with its factor graph shown in Figure 1.4. The sparsity patterns for the corresponding sparse Jacobian matrix  $\mathbf{A}$  is shown in Figure 1.9. Also shown is the pattern for the information matrix  $\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A}$ , in the top-right corner. On the right of Figure 1.9, we show the resulting upper triangular Choleksy factor  $\mathbf{R}$  for two different orderings. Both of them are sparse, and both of them satisfy  $\mathbf{R}^\top \mathbf{R} = \mathbf{A}^\top \mathbf{A}$  (up to a permutation of the variables), but they differ in the amount of sparsity they exhibit. It is exactly this that will determine how expensive it is to factorize  $\mathbf{A}$ . The first version of the ordering comes naturally: the poses come first and then the landmarks, leading to a sparse  $\mathbf{R}$  factor with 9399 nonzero entries. In contrast, the sparse factor  $\mathbf{R}$  in the bottom right was obtained by reordering the variables according to the Column approximate minimum degree permutation (COLAMD) heuristic [29, 241] and only has 4168 nonzero entries. Yet back-substitution gives exactly the same solution for both versions.

It is worth mentioning that other tools, like pre-conditioned conjugate gradient, can solve the normal equations *iteratively*. In visual SLAM, which has a very specific sparsity pattern, power iterations have also been used successfully [1168]. However, sparse factorization is still the method of choice for most SLAM problems and has a nice graphical model interpretation, which we discuss next.

## 1.6 Elimination

We have so far restricted ourselves to a linear-algebra explanation of performing inference for SLAM. In this section, we expand our worldview by thinking about inference more abstractly using graphical models directly. This will ultimately lead us to current state-of-the-art SLAM solvers based on a concept called the *Bayes tree* for incremental smoothing and mapping in the next section.

### 1.6.1 Variable Elimination Algorithm

There exists a general algorithm that can, given any (preferably sparse) factor graph, compute the corresponding posterior density  $p(\mathbf{x}|\mathbf{z})$  on the unknown variables  $\mathbf{x}$  in a form that allows easy recovery of the MAP solution to the problem. As we saw, a factor graph represents the unnormalized posterior  $\phi(\mathbf{x}) \propto p(\mathbf{x}|\mathbf{z})$  as a product of factors, and in SLAM problems this graph is typically generated directly from the measurements. The *variable elimination* algorithm is a recipe for converting a factor graph into another graphical model called a *Bayes net*, which depends only on the unknown variables  $\mathbf{x}$ . This then allows for easy MAP inference (as well as other operations such as sampling and/or marginalization).

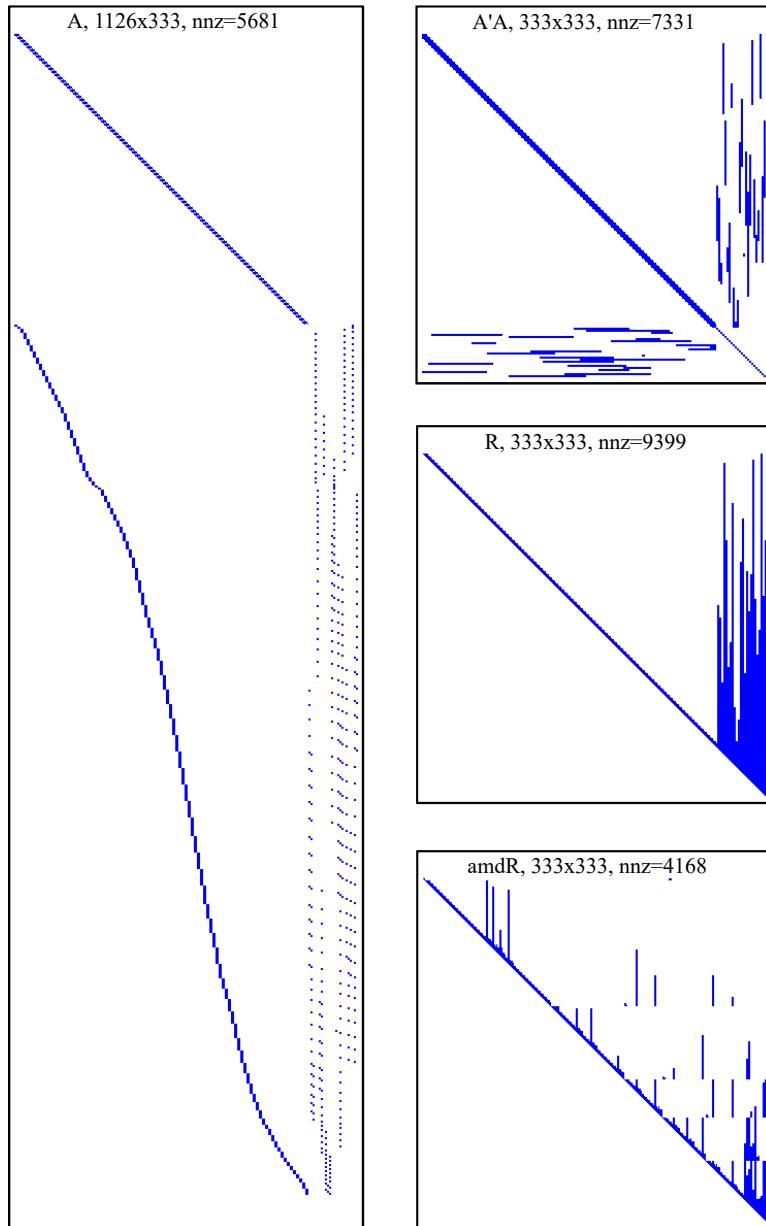


Figure 1.9 On the left, the measurement Jacobian  $\mathbf{A}$  associated with the problem in Figure 1.4, which has  $3 \times 95 + 2 \times 24 = 333$  unknowns. The number of rows, 1126, is equal to the number of (scalar) measurements. Also given is the number of nonzero entries “nnz”. On the right: (top) the information matrix  $\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A}$ ; (middle) its upper triangular Cholesky triangle  $\mathbf{R}$ ; (bottom) an alternative factor  $\text{amdR}$  obtained with a better variable ordering (COLAMD).

In particular, the variable elimination algorithm is a way to factorize any factor graph of the form

$$\phi(\mathbf{x}) = \phi(\mathbf{x}_1, \dots, \mathbf{x}_n) \quad (1.41)$$

into a factored Bayes net probability density of the form

$$p(\mathbf{x}) = p(\mathbf{x}_1|\mathbf{s}_1)p(\mathbf{x}_2|\mathbf{s}_2)\dots p(\mathbf{x}_n) = \prod_j p(\mathbf{x}_j|\mathbf{s}_j), \quad (1.42)$$

where  $\mathbf{s}_j$  denotes an assignment to the *separator*  $\mathbf{s}(\mathbf{x}_j)$  associated with variable  $\mathbf{x}_j$  under the chosen variable ordering  $\mathbf{x}_1, \dots, \mathbf{x}_n$ . The separator is defined as the set of variables on which  $\mathbf{x}_j$  is conditioned, after elimination. While this factorization is akin to the chain rule, eliminating a sparse factor graph will typically lead to small separators.

The elimination algorithm proceeds by eliminating one variable  $\mathbf{x}_j$  at a time, starting with the complete factor graph  $\phi_{1:n}$ . As we eliminate each variable  $\mathbf{x}_j$ , we generate a single conditional  $p(\mathbf{x}_j|\mathbf{s}_j)$ , as well as a reduced factor graph  $\phi_{j+1:n}$  on the remaining variables. After all variables have been eliminated, the algorithm returns the resulting Bayes net with the desired factorization.

To eliminate a single variable  $\mathbf{x}_j$  given a partially eliminated factor graph  $\phi_{j:n}$ , we first remove all factors  $\phi_i(\mathbf{x}_i)$  that are adjacent to  $\mathbf{x}_j$  and multiply them into the product factor  $\psi(\mathbf{x}_j, \mathbf{s}_j)$ . We then factorize  $\psi(\mathbf{x}_j, \mathbf{s}_j)$  into a conditional distribution  $p(\mathbf{x}_j|\mathbf{s}_j)$  on the eliminated variable  $\mathbf{x}_j$ , and a new factor  $\tau(\mathbf{s}_j)$  on the separator  $\mathbf{s}_j$ :

$$\psi(\mathbf{x}_j, \mathbf{s}_j) = p(\mathbf{x}_j|\mathbf{s}_j)\tau(\mathbf{s}_j). \quad (1.43)$$

Hence, *the entire factorization from  $\phi(\mathbf{x})$  to  $p(\mathbf{x})$  is seen to be a succession of  $n$  local factorization steps*. When eliminating the last variable  $\mathbf{x}_n$  the separator  $\mathbf{s}_n$  will be empty, and the conditional produced will simply be a prior  $p(\mathbf{x}_n)$  on  $\mathbf{x}_n$ .

One possible elimination sequence for the toy example is shown in Figure 1.10, for the ordering  $\ell_1, \ell_2, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ . In each step, the variable being eliminated is shaded gray, and the new factor  $\tau(\mathbf{s}_j)$  on the separator  $\mathbf{s}_j$  is shown in red. Taken as a whole, the variable elimination algorithm factorizes the factor graph  $\phi(\ell_1, \ell_2, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$  into the Bayes net in Figure 1.10 (bottom right), corresponding to the factorization

$$p(\ell_1, \ell_2, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = p(\ell_1|\mathbf{p}_1, \mathbf{p}_2)p(\ell_2|\mathbf{p}_3)p(\mathbf{p}_1|\mathbf{p}_2)p(\mathbf{p}_2|\mathbf{p}_3)p(\mathbf{p}_3). \quad (1.44)$$

### 1.6.2 Linear-Gaussian Elimination

In the case of linear measurement functions and additive normally distributed noise, the *elimination algorithm is equivalent to sparse matrix factorization*. Both sparse Cholesky and QR factorization are a special case of the general algorithm.

As explained before, the elimination algorithm proceeds one variable at a time. For every variable  $\mathbf{x}_j$  we remove all factors  $\phi_i(\mathbf{x}_i)$  adjacent to  $\mathbf{x}_j$  and form the

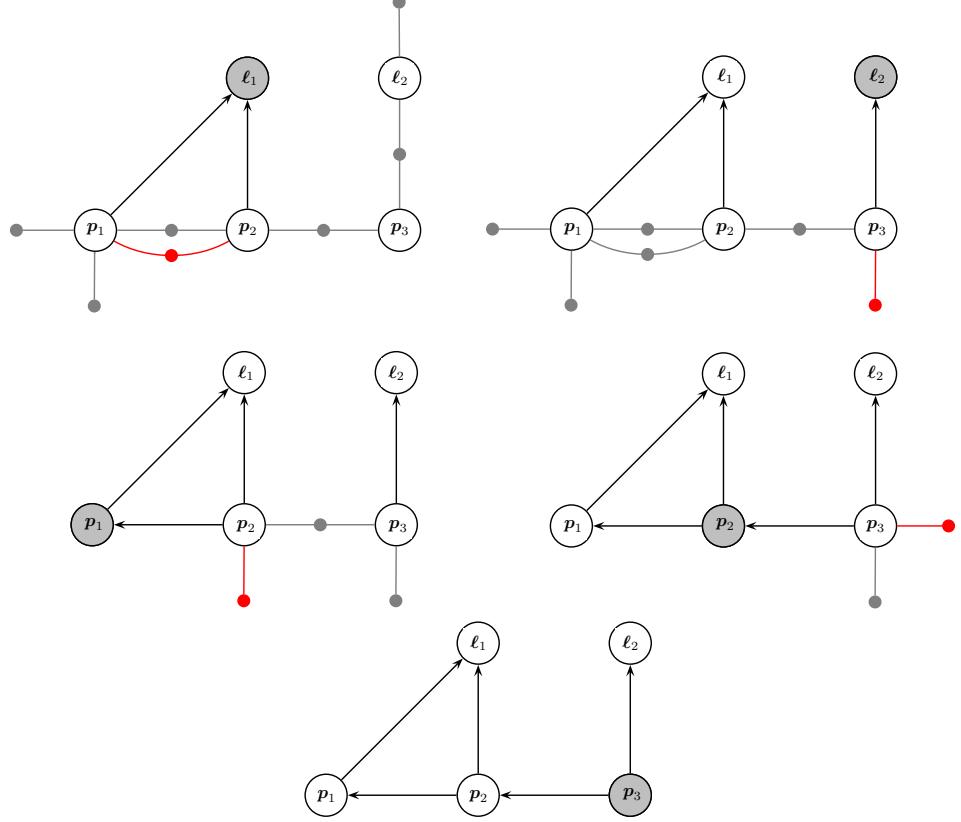


Figure 1.10 Variable elimination for the toy SLAM example, transforming the factor graph from Figure 1.2 into a Bayes net (bottom right), using the ordering  $\ell_1, \ell_2, p_1, p_2, p_3$ .

intermediate product factor  $\psi(\mathbf{x}_j, \mathbf{s}_j)$ . This can be done by accumulating all the matrices  $\mathbf{A}_i$  into a new, larger block-matrix  $\bar{\mathbf{A}}_j$ , as we can write

$$\psi(\mathbf{x}_j, \mathbf{s}_j) \leftarrow \prod_{i \in \mathcal{N}_j} \phi_i(\mathbf{x}_i) \quad (1.45a)$$

$$= \exp\left(-\frac{1}{2} \sum_i \|\mathbf{A}_i \mathbf{x}_i - \mathbf{b}_i\|_2^2\right) \quad (1.45b)$$

$$= \exp\left(-\frac{1}{2} \|\bar{\mathbf{A}}_j[\mathbf{x}_j; \mathbf{s}_j] - \bar{\mathbf{b}}_j\|_2^2\right), \quad (1.45c)$$

where the new RHS vector  $\bar{\mathbf{b}}_j$  stacks all  $\mathbf{b}_i$  and ‘;’ also denotes vertical stacking.

Consider eliminating the variable  $\ell_1$  in the toy example. The adjacent factors are  $\phi_4, \phi_7$ , and  $\phi_8$ , in turn inducing the separator  $\mathbf{s}_1 = [\mathbf{p}_1; \mathbf{p}_2]$ . The product factor is

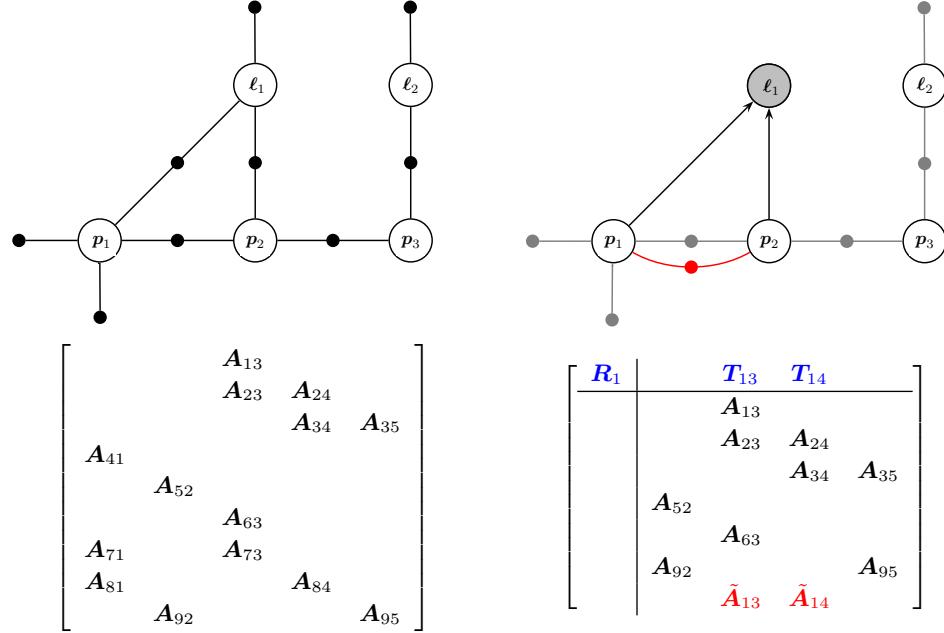


Figure 1.11 Eliminating the variable  $\ell_1$  as a partial sparse factorization step.

then equal to

$$\psi(\ell_1, p_1, p_2) = \exp\left(-\frac{1}{2}\|\bar{\mathbf{A}}_1[\ell_1; p_1; p_2] - \bar{\mathbf{b}}_1\|_2^2\right), \quad (1.46)$$

with

$$\bar{\mathbf{A}}_1 \triangleq \begin{bmatrix} \mathbf{A}_{41} & & \\ \mathbf{A}_{71} & \mathbf{A}_{73} & \\ \mathbf{A}_{81} & & \mathbf{A}_{84} \end{bmatrix}, \quad \bar{\mathbf{b}}_1 \triangleq \begin{bmatrix} \mathbf{b}_4 \\ \mathbf{b}_7 \\ \mathbf{b}_8 \end{bmatrix}. \quad (1.47)$$

Looking at the sparse Jacobian in Figure 1.6, this simply boils down to taking out the block-rows with nonzero blocks in the first column, corresponding to the three factors adjacent to  $\ell_1$ .

Next, factorizing the product  $\psi(\mathbf{x}_j, \mathbf{s}_j)$  can be done in several different ways. We discuss the QR variant, as it more directly connects to the linearized factors. In particular, the augmented matrix  $[\bar{\mathbf{A}}_j | \bar{\mathbf{b}}_j]$  corresponding to the product factor  $\psi(\mathbf{x}_j, \mathbf{s}_j)$  can be rewritten using partial QR-factorization [389] as follows:

$$[\bar{\mathbf{A}}_j | \bar{\mathbf{b}}_j] = \mathbf{Q} \begin{bmatrix} \mathbf{R}_j & \mathbf{T}_j & \mathbf{d}_j \\ & \tilde{\mathbf{A}}_\tau & \tilde{\mathbf{b}}_\tau \end{bmatrix}, \quad (1.48)$$

where  $\mathbf{R}_j$  is an upper-triangular matrix. This allows us to factor  $\psi(\mathbf{x}_j, \mathbf{s}_j)$  as follows:

$$\psi(\mathbf{x}_j, \mathbf{s}_j) = \exp \left\{ -\frac{1}{2} \|\bar{\mathbf{A}}_j[\mathbf{x}_j; \mathbf{s}_j] - \bar{\mathbf{b}}_j\|_2^2 \right\} \quad (1.49a)$$

$$= \exp \left\{ -\frac{1}{2} \|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2 \right\} \exp \left\{ -\frac{1}{2} \|\tilde{\mathbf{A}}_\tau \mathbf{s}_j - \tilde{\mathbf{b}}_\tau\|_2^2 \right\} \\ = p(\mathbf{x}_j | \mathbf{s}_j) \tau(\mathbf{s}_j), \quad (1.49b)$$

where we used the fact that the rotation matrix  $\mathbf{Q}$  does not alter the value of the norms involved.

In the toy example, Figure 1.11 shows the result of eliminating the first variable in the example, the landmark  $\ell_1$  with separator  $[\mathbf{p}_1; \mathbf{p}_2]$ . We show the operation on the factor graph *and* the corresponding effect on the sparse Jacobian from Figure 1.6, omitting the RHS. The partition above the line corresponds to a sparse, upper-triangular matrix  $\mathbf{R}$  that is being formed. New contributions to the matrix are shown: blue for the contributions to  $\mathbf{R}$ , and red for newly created factors. For completeness, we show the four remaining variable elimination steps in Figure 1.12, showing an end-to-end example of how QR factorization proceeds on a small example. The final step shows the equivalence between the resulting Bayes net and the sparse upper-triangular factor  $\mathbf{R}$ .

The entire elimination algorithm, using partial QR to eliminate a single variable, is equivalent to *sparse QR factorization*. As the treatment above considers multi-dimensional variables  $\mathbf{x}_j \in \mathbb{R}^{n_j}$ , this is in fact an instance of *multi-frontal QR factorization* [288], as we eliminate several scalar variables at a time, which is beneficial for processor utilization. While in our case the scalar variables are grouped because of their semantic meaning in the inference problem, sparse linear algebra codes typically analyze the problem to group for maximum computational efficiency. In many cases these two strategies are closely aligned.

*sparse QR factorization*  
*multi-frontal QR factorization*

### 1.6.3 Sparse Cholesky Factor as a Bayes Net

The equivalence between variable elimination and sparse matrix factorization reveals that the graphical model associated with an upper triangular matrix is a Bayes net! Just like a factor graph is the graphical embodiment of a sparse Jacobian, and an MRF can be associated with the Hessian, a Bayes net reveals the sparsity structure of a Cholesky factor. In hindsight, this perhaps is not too surprising: a Bayes net is a directed acyclic graph (DAG), and that is exactly the ‘upper-triangular’ property for matrices.

*MRF*

What’s more, the Cholesky factor corresponds to a *Gaussian Bayes net*, which we define as one made up of linear-Gaussian conditionals. The variable elimination algorithm holds for general densities, but in case the factor graph only contains linear measurement functions and Gaussian additive noise, the resulting Bayes net

*Gaussian Bayes net*

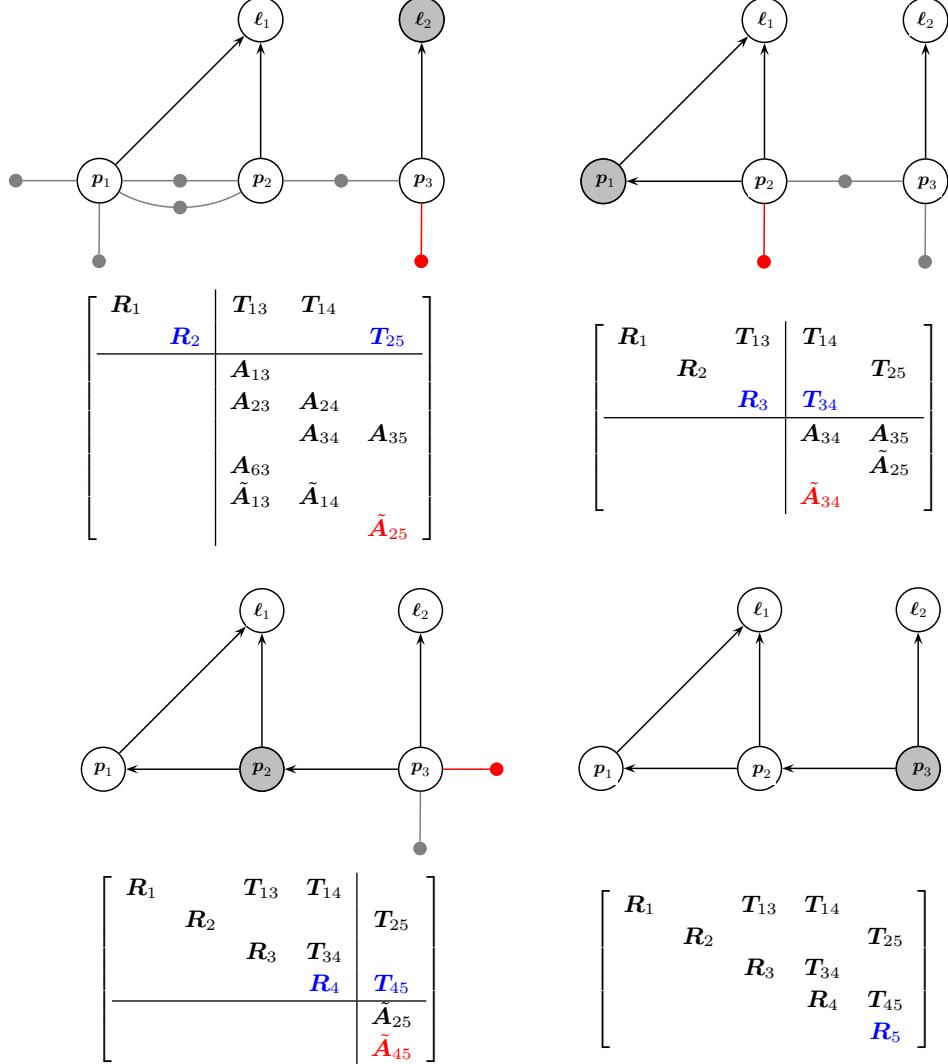


Figure 1.12 The remaining elimination steps for the toy example, completing a full QR factorization. The last step in the bottom right shows the equivalence between the resulting Bayes net and the sparse Cholesky factor  $\mathbf{R}$ .

has a very specific form. We discuss the details below, as well as how to solve for the MAP estimate in the linear case.

As we discussed, the Gaussian factor graph corresponding to the linearized non-linear problem is transformed by elimination into the density  $p(\mathbf{x})$  given by the

now-familiar Bayes-net factorization:

$$p(\mathbf{x}) = \prod_j p(\mathbf{x}_j | \mathbf{s}_j). \quad (1.50)$$

In both QR and Cholesky variants, the conditional densities  $p(\mathbf{x}_j | \mathbf{s}_j)$  are given by

$$p(\mathbf{x}_j | \mathbf{s}_j) = k \exp\left(-\frac{1}{2} \|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2\right), \quad (1.51)$$

which is a linear-Gaussian density on the eliminated variable  $\mathbf{x}_j$ . Indeed, we have

$$\|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2 = (\mathbf{x}_j - \boldsymbol{\mu}_j)^\top \mathbf{R}_j^\top \mathbf{R}_j (\mathbf{x}_j - \boldsymbol{\mu}_j) \triangleq \|\mathbf{x}_j - \boldsymbol{\mu}_j\|_{\boldsymbol{\Sigma}_j}^2, \quad (1.52)$$

where the mean  $\boldsymbol{\mu}_j = \mathbf{R}_j^{-1}(\mathbf{d}_j - \mathbf{T}_j \mathbf{s}_j)$  depends linearly on the separator  $\mathbf{s}_j$ , and the covariance matrix is given by  $\boldsymbol{\Sigma}_j = (\mathbf{R}_j^\top \mathbf{R}_j)^{-1}$ . Hence, the normalization constant  $k = |2\pi \boldsymbol{\Sigma}_j|^{-\frac{1}{2}}$ .

After the elimination step is complete, back-substitution is used to obtain the MAP estimate of each variable. As seen in Figure 1.12, the last variable eliminated does not depend on any other variables. Thus, the MAP estimate of the last variable can be directly extracted from the Bayes net. By proceeding in reverse elimination order, the values of all the separator variables for each conditional will always be available from the previous steps, allowing the estimate for the current frontal variable to be computed.

At every step, the MAP estimate for the variable  $\mathbf{x}_j$  is the conditional mean,

$$\mathbf{x}_j^* = \mathbf{R}_j^{-1}(\mathbf{d}_j - \mathbf{T}_j \mathbf{s}_j^*), \quad (1.53)$$

since by construction the MAP estimate for the separator  $\mathbf{s}_j^*$  is fully known by this point.

## 1.7 Incremental SLAM

In an incremental SLAM setting, we want to compute the optimal trajectory and map whenever we receive new measurements while traversing the environment, or at least at regular intervals. One way to do so is to *update* the most recent matrix factorization with the new measurements, to reuse the computation that already incorporated all previous measurements. In the linear case, this is possible through incremental factorization methods, the dense versions of which are also discussed at length in Golub and Loan [389]. However, matrix factorization operates on linear systems, but most SLAM problems of practical interest are *nonlinear*. Using incremental matrix factorization, it is far from obvious how re-linearization can be performed incrementally without refactoring the complete matrix. To overcome this problem we once again resort to graphical models, and introduce a new graphical model, the *Bayes tree*. We then show how to incrementally update the Bayes tree as

*Bayes tree*

new measurements and states are added to the system, leading to the incremental smoothing and mapping (iSAM) algorithm.

### 1.7.1 The Bayes Tree

It is well known that inference in a tree-structured graph is efficient. In contrast, the factor graphs associated with typical robotics problems contain many loops. Still, we can construct a tree-structured graphical model in a two-step process: first, perform variable elimination on the factor graph to obtain a Bayes net with a special property. Second, exploit that special property to find a tree structure over *cliques* in this Bayes net.

In particular, a Bayes net obtained by running the elimination algorithm on a factor graph satisfies a special property: it is *chordal*, meaning that any undirected cycle of length greater than three has a *chord*, i.e., an edge connecting two non-consecutive vertices on the cycle. In AI and machine learning, a chordal graph is more commonly said to be *triangulated*. Because it is still a Bayes net, the corresponding joint density  $p(\mathbf{x})$  is given by factorizing over the individual variables  $\mathbf{x}_j$ ,

$$p(\mathbf{x}) = \prod_j p(\mathbf{x}_j | \boldsymbol{\pi}_j), \quad (1.54)$$

where  $\boldsymbol{\pi}_j$  are the parent nodes of  $\mathbf{x}_j$ . However, although the Bayes net is chordal, at this variable level it is still a non-trivial graph: neither chain-like nor tree-structured. The chordal Bayes net for our running toy SLAM example is shown in the last step of Figure 1.10, and it is clear that there is an undirected cycle  $\mathbf{p}_1 - \mathbf{p}_2 - \ell_1$ , implying it does not have a tree-structured form.

By identifying cliques in this chordal graph, the Bayes net may be rewritten as a *Bayes tree*. We introduce this new, tree-structured graphical model to capture the *clique structure* of the Bayes net. It is not obvious that cliques in the Bayes net should form a tree. They do so because of the chordal property, although we will not attempt to prove that here. Listing all these cliques in an undirected tree yields a *clique tree*, also known as a *junction tree* in AI and machine learning. The Bayes tree is just a directed version of this that preserves information about the elimination order.

More formally, a Bayes tree is a directed tree where the nodes represent *cliques*  $\mathbf{c}_k$  of the underlying chordal Bayes net. In particular, we define one conditional density  $p(\mathbf{f}_k | \mathbf{s}_k)$  per node, with the *separator*  $\mathbf{s}_k$  as the intersection  $\mathbf{c}_k \cap \boldsymbol{\varpi}_k$  of the clique  $\mathbf{c}_k$  and its parent clique  $\boldsymbol{\varpi}_k$ . The *frontal variables*  $\mathbf{f}_k$  are the remaining variables, i.e.,  $\mathbf{f}_k \triangleq \mathbf{c}_k \setminus \mathbf{s}_k$ . Notationally, we write  $\mathbf{c}_k = \mathbf{f}_k : \mathbf{s}_k$  for a clique. The following expression gives the joint density  $p(\mathbf{x})$  on the variables  $\mathbf{x}$  defined by a Bayes tree:

$$p(\mathbf{x}) = \prod_k p(\mathbf{f}_k | \mathbf{s}_k). \quad (1.55)$$

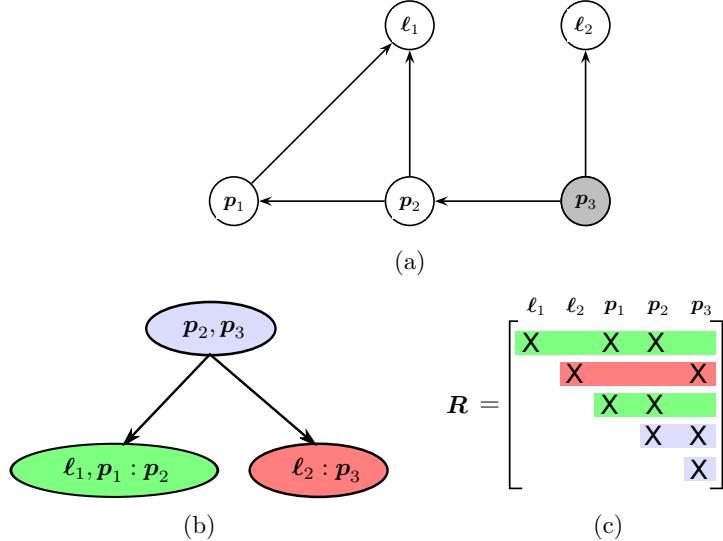


Figure 1.13 The Bayes tree (b) and the associated square root information matrix  $\mathbf{R}$  (c) describing the clique structure in the chordal Bayes net (a) based on our canonical example from Figure 1.2. A Bayes tree is similar to a clique tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques with rows in the  $\mathbf{R}$  factor is indicated by color.

For the root  $\mathbf{f}_r$ , the separator is empty, i.e., it is a simple prior  $p(\mathbf{f}_r)$  on the root variables. The way Bayes trees are defined, the separator  $\mathbf{s}_k$  for a clique  $\mathbf{c}_k$  is always a subset of the parent clique  $\varpi_k$ , and hence the directed edges in the graph have the same semantic meaning as in a Bayes net: conditioning.

The Bayes tree associated with our canonical toy SLAM problem (Figure 1.2) is shown in Figure 1.13. The root clique  $\mathbf{c}_1 = \mathbf{p}_2, \mathbf{p}_3$  (shown in blue) comprises  $\mathbf{p}_2$  and  $\mathbf{p}_3$ , which intersects with two other cliques,  $\mathbf{c}_2 = \ell_1, \mathbf{p}_1 : \mathbf{p}_2$  shown in green, and  $\mathbf{c}_3 = \ell_2 : \mathbf{p}_3$  shown in red. The colors also indicate how the rows of square-root information matrix  $\mathbf{R}$  map to the different cliques, and how the Bayes tree captures independence relationships between them. For example, the green and red rows only intersect in variables that belong to the root clique, as predicted.

### 1.7.2 Updating the Bayes Tree

Incremental inference corresponds to a simple editing of the Bayes tree. This view provides a better explanation and understanding of the otherwise abstract incremental matrix factorization process. It also allows us to store and compute the square-root information matrix in the form of a Bayes tree, a deeply meaningful sparse storage scheme.

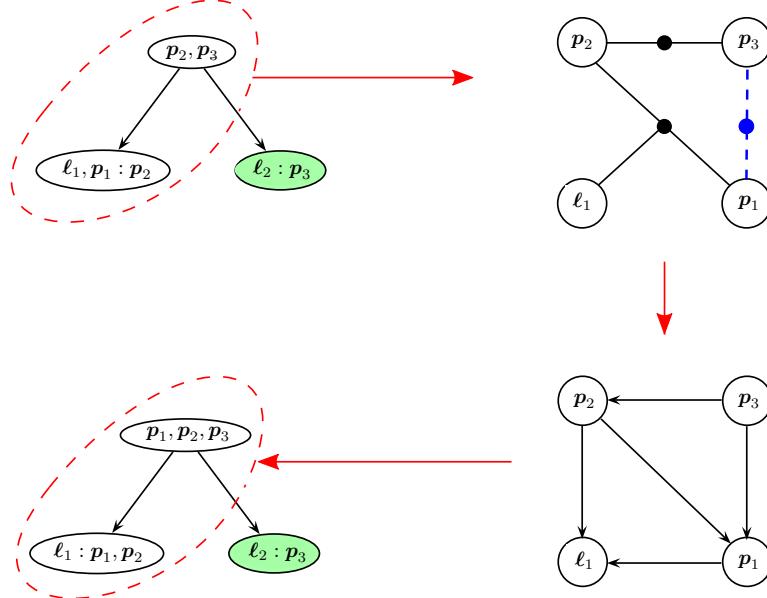


Figure 1.14 Updating a Bayes tree with a new factor, based on the example in Figure 1.13. The affected part of the Bayes tree is highlighted for the case of adding a new factor between  $p_1$  and  $p_3$ . Note that the right branch (green) is not affected by the change. (top right) The factor graph generated from the affected part of the tree with the new factor (dashed blue) inserted. (bottom right) The chordal Bayes net resulting from eliminating the factor graph. (bottom left) The Bayes tree created from the chordal Bayes net, with the unmodified right ‘orphan’ sub-tree from the original Bayes tree added back in.

To incrementally update the Bayes tree, we selectively convert part of the Bayes tree back into factor-graph form. When a new measurement is added this corresponds to adding a factor, e.g., a measurement involving two variables will induce a new binary factor  $\phi(\mathbf{x}_j, \mathbf{x}_{j'})$ . In this case, *only* the paths in the Bayes tree between the cliques containing  $\mathbf{x}_j$  and  $\mathbf{x}_{j'}$  and the root will be affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing  $\mathbf{x}_j$  or  $\mathbf{x}_{j'}$ . Hence, to update the Bayes tree, the affected parts of the tree are converted back into a factor graph, and the new factor associated with the new measurement is added to it. By re-eliminating this temporary factor graph, using whatever elimination ordering is convenient, a new Bayes tree is formed and the unaffected sub-trees can be reattached.

In order to understand why only the top part of the tree is affected, we look at two important properties of the Bayes tree. These directly arise from the fact that it encodes the information flow during elimination. The Bayes tree is formed from the chordal Bayes net following the inverse elimination order. In this way, variables in each clique collect information from their child cliques via the elimination of

these children. Thus, information in any clique propagates only upwards to the root. Second, the information from a factor enters elimination only when the first variable connected to that factor is eliminated. Combining these two properties, we see that a new factor cannot influence any other variables that are not successors of the factor's variables. However, a factor involving variables having different (i.e., independent) paths to the root means that these paths must now be re-eliminated to express the new dependency between them.

Figure 1.14 shows how these incremental factorization/inference steps are applied to our canonical SLAM example. In this example, we add a new factor between  $\mathbf{p}_1$  and  $\mathbf{p}_3$ , affecting only the left branch of the tree, marked by the red dashed line in to top-left figure. We then create the factor graph shown in the top-right figure by creating a factor for each of the clique densities,  $p(\mathbf{p}_2, \mathbf{p}_3)$  and  $p(\ell_1, \mathbf{p}_1 | \mathbf{p}_2)$ , and add the new factor  $f(\mathbf{p}_1, \mathbf{p}_3)$ . The bottom-right figure shows the eliminated graph using the ordering  $\ell_1, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ . And finally, in the bottom-left figure, the reassembled Bayes tree is shown consisting of two parts: the Bayes tree derived from the eliminated graph, and the unaffected clique from the original Bayes tree (shown in green).

Figure 1.15 shows an example of the Bayes tree for a small SLAM sequence. Shown is the tree for step 400 of the well-known Manhattan world simulated sequence by Olson et al. [821]. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated.

### 1.7.3 Incremental Smoothing and Mapping

Putting all of the above together and addressing some practical considerations about re-linearization yields a state-of-the-art incremental, nonlinear approach to MAP estimation in robotics, iSAM. The first version, iSAM1[530], used the incremental matrix factorization methods from Golub and Loan [389]. However, linearization in iSAM1 was handled in a sub-optimal way: it was done for the full factor graph at periodic instances and/or when matrix fill-in became unwieldy. The second version of the approach, iSAM2, uses a Bayes tree representation for the posterior density [533]. It then employs Bayes tree incremental updating as each new measurement comes in, as described above.

*What variable ordering should we use in re-eliminating the affected cliques?* Only the variables in the affected part of the Bayes tree are updated. One strategy then is to apply COLAMD locally to the affected variables. However, we can do better: we force recently accessed variables to the end of the ordering, i.e., into the root clique. For this incremental variable ordering strategy one can use the constrained COLAMD algorithm [241]. This both forces the most recently accessed variables to the end and still provides a good overall ordering. Generally, subsequent updates will then only affect a small part of the tree, and can therefore be expected to be efficient in most cases, except for large loop closures.

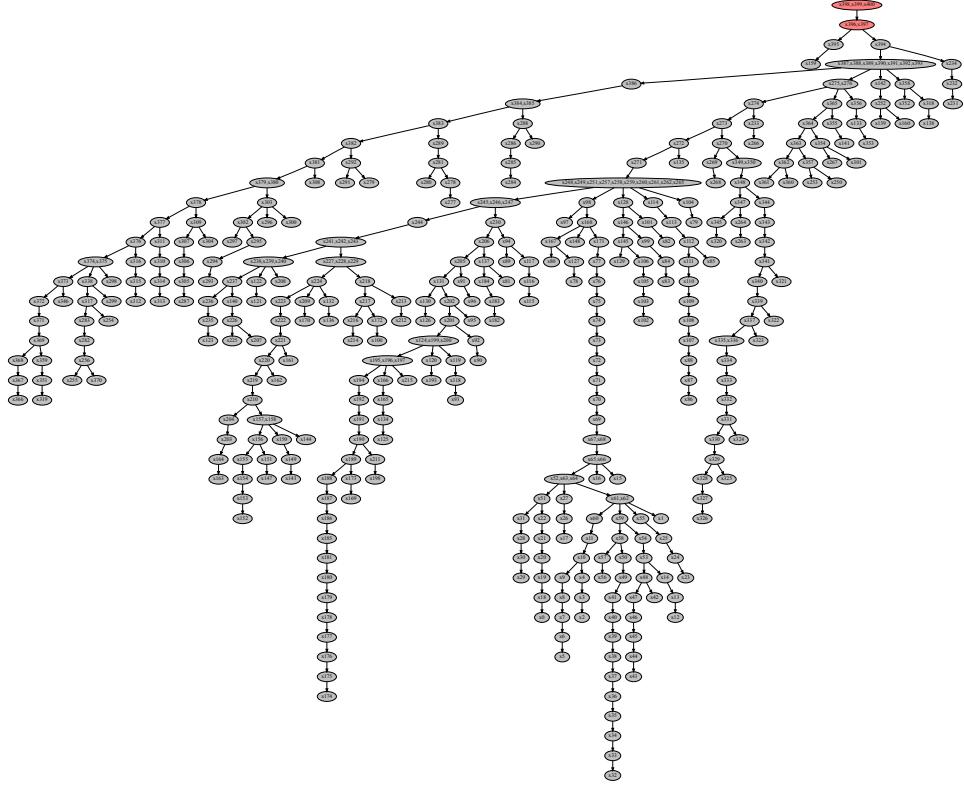


Figure 1.15 An example of the Bayes tree data structure for a small SLAM sequence. The incremental nonlinear least squares estimation algorithm iSAM2 [533] is based on viewing incremental factorization as editing the graphical model corresponding to the posterior probability of the solution, the Bayes tree. As a robot explores the environment, new measurements often only affect small parts of the tree, and only those parts are recalculated (shown in red).

After updating the tree we also need to *update the solution*. Back-substitution in the Bayes tree proceeds from the root (which does not depend on any other variables) and proceeds to the leaves. However, it is typically not necessary to recompute a solution for all variables: local updates to the tree often do not affect variables in remote parts of the tree. Instead, at each clique we can check the difference in variable estimates that is propagated downwards and stop when this difference falls below a small threshold.

Our motivation for introducing the Bayes tree was to incrementally solve nonlinear optimization problems. For this we *selectively re-linearize* factors that contain variables whose deviation from the linearization point exceeds a small threshold. In contrast to the tree modification above, we now have to redo all cliques that contain

the affected variables, not just as frontal variables, but also as separator variables. This affects larger parts of the tree, but in most cases is still significantly cheaper than recomputing the complete tree. We also have to go back to the original factors, instead of directly turning the cliques into a factor graph. And that requires caching certain quantities during elimination. The overall incremental nonlinear algorithm, iSAM2, is described in much more detail in [533].

iSAM1 and iSAM2 have been applied successfully to many different robotics estimation problems with non-trivial constraints between variables that number into the millions, as will be discussed subsequent chapters. Both are implemented in the GTSAM library, which can be found at <https://github.com/borglab/gtsam>.

### 1.8 Further Readings & Recent Trends

For those readers interested in learning more details about factor graphs, we recommend the longer article by Dellaert et al. [255]. Out of necessity, this chapter is somewhat brief and we have downplayed the more advanced concepts so as to keep the barrier to entry as low as possible. However, factor graphs and the general elimination algorithm are quite powerful and worth knowing in some detail. After this, we recommend looking at Kaess et al. [533] to better understand the concept of the Bayes tree, which underlies modern solvers such as GTSAM.

Today, the factor graph paradigm can be used to handle state variables on manifolds, incorporate many different kinds of sensors, made to handle outlier measurements, and can even fold in the outputs of deep learning methods. Much of the rest of this handbook discusses these trends and so to learn more, keep reading.

# 2

## Advanced State Variable Representations

Timothy Barfoot, Frank Dellaert, Michael Kaess, and Jose Luis Blanco-Claraco

The previous chapter detailed how to set up and solve a SLAM problem using the factor-graph paradigm. We deliberately avoided discussing some subtleties of the state variables we were estimating. In this chapter, we revisit the nature of our state variables and introduce two important topics that are prevalent in modern SLAM formulations. First and foremost, we need some better tools for handling state variables that have certain constraints associated with them; these constraints define a *manifold* for our variables, which subsequently then require special care during optimization. There are many examples of manifolds that appear in SLAM, the most common being those associated with the rotational aspects of a robot (especially in three dimensions, but even in the plane). A second aspect of state variables stems from the nature of time itself. In the previous chapter, we implicitly assumed that our robot moved in discrete-time steps through the world. In this chapter we introduce smooth, *continuous-time* representations of trajectories and discuss how these are fully compatible with our factor-graph formulation. We use Barfoot [47] as the primary reference with some streamlined notation from Sola et al. [1019].

### 2.1 Optimization on Manifolds

*manifold*

While in some robotics problems we can get away with vector-valued unknowns, in most practical situations we have to deal with three-dimensional rotations and other non-vector manifolds. Loosely speaking, a manifold is a topologically closed surface (*e.g.*, the perimeter of a circle, or the surface of a sphere); importantly, a manifold resembles Euclidean space locally near each point. Manifolds require a more sophisticated machinery that takes into account their special structure. In this section, we discuss how to perform optimization on manifolds, which will build upon the optimization framework for vector spaces from the previous chapter. As an example, Figure 2.1 visualizes a spherical manifold,  $\mathcal{M}$ , and its tangent space,  $T_\chi\mathcal{M}$ , which can be used as a local coordinate system at  $\chi \in \mathcal{M}$  for optimization.

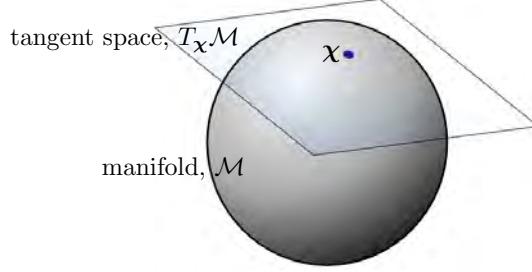


Figure 2.1 For the sphere manifold,  $\mathcal{M}$ , the local tangent plane,  $T_x \mathcal{M}$ , with a local basis provides the notion of local coordinates.

### 2.1.1 Rotations and Poses

While there are several manifolds that can be discussed in the context of SLAM, the two most common are those used to represent rotations and poses. Rotations are typically either in two (planar) or three dimensions and we therefore refer to the manifold of rotations as the *special orthogonal group*  $\text{SO}(d)$ , where  $d = 2$  or  $3$ , accordingly. A planar *rotation matrix*,  $\mathbf{R}_a^b \in \text{SO}(2)$ , has the form

*special orthogonal group  
rotation matrix*

$$\mathbf{R}_a^b = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad (2.1)$$

where  $\theta \in \mathbb{R}$ , the angle of rotation, is the single degree of freedom in this case. Moreover,  $\mathbf{R}_a^b$  allows us to rotate a two-dimensional vector (*i.e.*, landmark) expressed in reference frame  $\mathcal{F}^a$  to  $\mathcal{F}^b$ :  $\ell^b = \mathbf{R}_a^b \ell^a$ .

A rotation matrix in three dimensions,  $\mathbf{R}_a^b \in \text{SO}(3)$ , again rotates vectors (this time in three dimensions) from one frame to another. Three-dimensional rotation matrices have nine entries but only three degrees of freedom (*e.g.*, roll, pitch, yaw). Both two- and three-dimensional rotation matrices must satisfy the constraints  $\mathbf{R}_a^{b\top} \mathbf{R}_a^b = \mathbf{I}$  and  $\det(\mathbf{R}_a^b) = 1$  to limit their degrees of freedom appropriately.

The *pose* of a robot comprises both rotational,  $\mathbf{R}_a^b \in \text{SO}(d)$ , and translational,  $\mathbf{t}_a^b \in \mathbb{R}^d$ , variables with  $3(d - 1)$  degrees of freedom in all. Sometimes we keep track of these quantities separately and then can use  $\{\mathbf{R}_a^b, \mathbf{t}_a^b\} \in \text{SO}(d) \times \mathbb{R}^d$  as the representation. Alternatively, these quantities can be assembled into a  $(d+1) \times (d+1)$  *transformation matrix*,

$$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}. \quad (2.2)$$

The manifold of all such transformation matrices is called the *special Euclidean group*,  $\text{SE}(d)$ , where again  $d = 2$  (planar motion) or  $3$  (three-dimensional motion). The benefit of using  $\text{SE}(d)$  is that we can easily translate and rotate landmarks

*special Euclidean group  
matrix*

using a single matrix multiplication:

$$\underbrace{\begin{bmatrix} \ell^b \\ 1 \end{bmatrix}}_{\tilde{\ell}^b} = \underbrace{\begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathbf{T}_a^b} \underbrace{\begin{bmatrix} \ell^a \\ 1 \end{bmatrix}}_{\tilde{\ell}^a}, \quad (2.3)$$

*homogeneous point*

*Lie groups*

Due to the constraints imposed on the forms of rotation and transformation matrices, they are unfortunately not vectors. For example, we cannot simply add two rotation matrices together and arrive at another valid rotation matrix. However, it turns out that  $\text{SO}(d)$  and  $\text{SE}(d)$  are examples of manifolds that possess some extra useful properties, called *matrix Lie groups*. We can exploit the structure of these manifolds to continue to perform unconstrained MAP optimization for factor-graph SLAM (see, for example, Dellaert et al. [255], Boumal [106], or Barfoot [47]). For additional background on Lie groups in robotics see the seminal work of Chirikjian and Kyatkin [207], Chirikjian [205, 206].

### 2.1.2 Matrix Lie Groups

The key to performing optimization on  $\text{SO}(d)$  and  $\text{SE}(d)$  is to exploit their group structure. For example, one nice property is that matrix Lie groups enjoy *closure* under matrix multiplication, so that if we multiply two members, e.g.,  $\mathbf{R}_b^c, \mathbf{R}_a^b \in \text{SO}(d)$ , the result is also in the group:  $\mathbf{R}_a^c = \mathbf{R}_b^c \mathbf{R}_a^b \in \text{SO}(d)$ .

*Lie algebra*

Another nice property of matrix Lie groups is that they come along with a very useful companion structure called a *Lie algebra*, which is also the tangent space for the Lie group at the identity element. For our purposes, the most important aspects of the Lie algebra are (i) that it comprises a vector space with dimension equal to the number of degrees of freedom of its Lie group, and (ii) there is a well-established mapping (the matrix exponential) from the Lie algebra to the Lie group. This allows us to construct elements of the Lie group with relative ease from elements of the Lie algebra. For example, for  $\text{SO}(2)$  we can build a rotation matrix (dropping super/subscripts for now) according to

$$\mathbf{R} = \text{Exp}(\theta) = \exp(\theta^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} (\theta^\wedge)^n = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \in \text{SO}(2), \quad (2.4)$$

where

$$\theta^\wedge = \begin{bmatrix} 0 & -\theta \\ \theta & 0 \end{bmatrix}, \quad \theta \in \mathbb{R}. \quad (2.5)$$

As summarized in Figure 2.2 along with the rest of the operators introduced in this chapter, the quantity  $\theta^\wedge$  is a member of the Lie algebra,  $\text{so}(2)$ , and it is mapped through the matrix exponential,  $\exp(\cdot)$ , to a member of the Lie group,  $\mathbf{R}$ . We can go the other way with the matrix logarithm:  $\theta = \text{Log}(\mathbf{R}) = (\log(\mathbf{R}))^\vee$ , where we

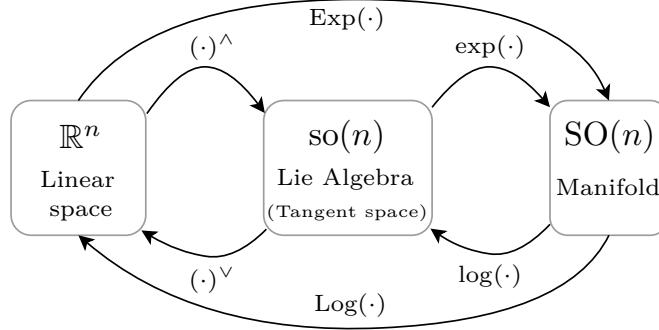


Figure 2.2 Summary of notation introduced in this chapter: the *wedge*  $(\cdot)^\wedge$  and *vee*  $(\cdot)^\vee$  operators, the exponential  $\exp(\cdot)$  and logarithm  $\log(\cdot)$  maps, which often are just the matrix exponential and logarithm, and the convenience shortcut operators  $\text{Exp}(\cdot) = \exp((\cdot)^\wedge)$  and  $\text{Log}(\cdot) = \log((\cdot)^\vee)$ . The  $\text{SO}(n)$  manifold is used for illustrative purposes but operators are defined for any other manifold.

use the matrix logarithm and the *vee* operator,  $(\cdot)^\vee$ , which is the inverse of the *wedge* operator,  $(\cdot)^\wedge$ .

Each matrix Lie group has its own linear  $(\cdot)^\wedge$  operator used to construct a Lie algebra member from the standard vector space of appropriate dimension. For  $\text{SO}(3)$ , it is the skew-symmetric operator:

$$\mathbf{R} = \text{Exp}(\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{\theta}^{\wedge n} \in \text{SO}(3), \quad (2.6a)$$

$$\boldsymbol{\theta}^\wedge = \begin{bmatrix} 0 & -\theta_3 & \theta_2 \\ \theta_3 & 0 & -\theta_1 \\ -\theta_2 & \theta_1 & 0 \end{bmatrix} \in \text{so}(3), \quad \boldsymbol{\theta} = \begin{bmatrix} 0 & -\theta_3 & \theta_2 \\ \theta_3 & 0 & -\theta_1 \\ -\theta_2 & \theta_1 & 0 \end{bmatrix}^\vee = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \in \mathbb{R}^3. \quad (2.6b)$$

For  $\text{SE}(d)$ , we can use

$$\mathbf{T} = \text{Exp}(\boldsymbol{\xi}) = \exp(\boldsymbol{\xi}^\wedge) \in \text{SE}(d), \quad (2.7a)$$

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\theta}^\wedge & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} \in \text{se}(d), \quad \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^{3(d-1)}, \quad \boldsymbol{\theta} \in \mathbb{R}^{2d-3}, \quad \boldsymbol{\rho} \in \mathbb{R}^d, \quad (2.7b)$$

where  $d = 2$  (planar) or  $3$  (three-dimensional). Note, the version of the  $(\cdot)^\wedge$  operator can be determined by the size of the input vector.

As we saw for  $\text{SO}(2)$  above, for each of the matrix Lie groups discussed here, there are also well-known closed-form expressions for the mappings between the Lie algebra and the Lie group that can be used rather than the infinite series form of the matrix exponential [47].

### 2.1.3 Lie Group Optimization

Now that we have these matrix Lie groups established, we can use them to help ‘linearize’ our nonlinear least-squares terms in order to carry out MAP inference. Looking back to the discussion in Section 1.3.1, we still seek to linearize our measurement functions,  $\mathbf{h}_i(\cdot)$ , only now the input to these may involve a member of a Lie group.

For example, suppose  $\mathbf{h}_i(\cdot)$  represents a camera model that takes as its input a homogeneous landmark expressed in the camera frame,  $\tilde{\ell}_i^c$ , and returns the pixel coordinates of the landmark in an image,  $\mathbf{z}_i \in \mathbb{R}^2$ :  $\mathbf{z}_i = \mathbf{h}_i(\tilde{\ell}_i^c)$ . We can write the generative sensor model therefore as

$$\mathbf{z}_i = \mathbf{h}_i \left( \mathbf{T}_w^c \tilde{\ell}_i^w \right) + \boldsymbol{\eta}_i, \quad (2.8)$$

where  $\mathbf{T}_w^c \in \text{SE}(3)$  is the pose of world frame with respect to the camera,  $\tilde{\ell}_i^w$  is the homogeneous landmark expressed in the world frame, and  $\boldsymbol{\eta}_i$  is the usual sensor noise. We then might like to solve the optimization problem

$$\mathbf{T}_w^{c^*} = \arg \min_{\mathbf{T}_w^c} = \sum_i \left\| \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}_w^c \tilde{\ell}_i^w \right) \right\|_{\Sigma_i}^2, \quad (2.9)$$

which is known as the perspective-n-point (PNP) problem. For this example, we assume that the positions of the landmarks in the world frame are known but of course in SLAM we might like to estimate these as well.

To linearize our sensor model, we use the fact that we can produce a perturbed version of our pose through its Lie algebra according to<sup>1</sup>

$$\mathbf{T}_w^c = \mathbf{T}_w^{c^0} \text{Exp}(\boldsymbol{\xi}). \quad (2.10)$$

Here,  $\boldsymbol{\xi} \in \mathbb{R}^6$  is used to produce a ‘small’ pose change that perturbs an initial guess,  $\mathbf{T}_w^{c^0} \in \text{SE}(3)$ . This perturbation is also sometimes written succinctly using the  $\oplus$  operator so that

$$\mathbf{T}_w^c = \mathbf{T}_w^{c^0} \oplus \boldsymbol{\xi} \quad (2.11)$$

is a shorthand for (2.10). Owing to the closure property discussed earlier, the product of these two quantities is guaranteed to be in  $\text{SE}(3)$ . By using the Lie algebra to define our pose perturbation, we restrict its dimension to be equal to the actual number of degrees of freedom in a three-dimensional pose, which will mean that we can avoid introducing constraints during optimization.

We can also approximate the perturbed pose according to

$$\mathbf{T}_w^c \approx \mathbf{T}_w^{c^0} (\mathbf{I} + \boldsymbol{\xi}^\wedge), \quad (2.12)$$

where we have kept just the terms up to linear in  $\boldsymbol{\xi}$  from the series form of the matrix

<sup>1</sup> It is also possible to perturb on the left side rather than the right. A more subtle question is whether the perturbation is happening on the ‘sensor’ side or the ‘world’ side, which depends on whether the unknown transform is  $\mathbf{T}_w^c$  or  $\mathbf{T}_c^w$  and whether the perturbation is applied to the left or the right.

exponential; see (2.6). Then, inserting (2.12) into our measurement function (2.8), we have

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} (\mathbf{I} + \boldsymbol{\xi}^\wedge) \tilde{\boldsymbol{\ell}}_i^w \right) + \boldsymbol{\eta}_i. \quad (2.13)$$

This can also be rewritten as

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w + \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \odot \boldsymbol{\xi} \right) + \boldsymbol{\eta}_i, \quad (2.14)$$

where  $\odot$  is a (linear) operator for homogeneous points [47]:

$$\tilde{\boldsymbol{\ell}}^\odot = \begin{bmatrix} \boldsymbol{\ell} \\ 1 \end{bmatrix}^\odot = \begin{bmatrix} \mathbf{I} & -\boldsymbol{\ell}^\wedge \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \quad (2.15)$$

We have essentially ‘linearized’ the pose perturbation in (2.14) and now need to linearize the camera function  $\mathbf{h}_i(\cdot)$  as well. We can use a standard first-order Taylor series approximation to write

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \right) + \underbrace{\frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\ell}} \Big|_{\mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w} \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \odot \boldsymbol{\xi} + \boldsymbol{\eta}_i}_{\mathbf{H}_i \text{ (chain rule)}}, \quad (2.16)$$

where the chaining of two pieces into the overall Jacobian,  $\mathbf{H}_i = \frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\xi}}$ , is now clear.

An alternative derivation for the same Jacobian can be reached by following a step-by-step decomposition of the sensor model into elementary operations (sensor model, pose composition, exponential operator, etc.) using stacked versions of matrices as intermediary vector states, as described in Blanco [87]. For this particular example, the Jacobian of the camera model with respect to the camera pose can be split into the product of four terms using the chain rule:

$$\mathbf{H}_i = \frac{\partial \mathbf{h}_i}{\partial \boldsymbol{\xi}} = \underbrace{\frac{\partial \mathbf{h}(\tilde{\boldsymbol{\ell}})}{\partial \tilde{\boldsymbol{\ell}}} \Big|_{\tilde{\boldsymbol{\ell}} = \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w}}_{\text{sensor model}} \cdot \underbrace{\frac{\partial \mathbf{T} \tilde{\boldsymbol{\ell}}}{\partial \mathbf{T}} \Big|_{\tilde{\boldsymbol{\ell}} = \tilde{\boldsymbol{\ell}}_i^w}}_{\text{relative point transformation}} \cdot \underbrace{\frac{\partial \mathbf{T}_1 \oplus \mathbf{T}_2}{\partial \mathbf{T}_2} \Big|_{\substack{\mathbf{T}_1 = \mathbf{T}_w^{c^0} \\ \mathbf{T}_2 = \text{Exp}(\boldsymbol{\xi}) = \mathbf{I}}}}_{\text{two pose composition}} \cdot \underbrace{\frac{\partial \text{Exp}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \Big|_{\boldsymbol{\xi} = \mathbf{0}}}_{\text{exponential map}}, \quad (2.17)$$

where each individual Jacobian has a known expression [87]. Note how the linearization point for each Jacobian takes into account that the perturbation happens around  $\boldsymbol{\xi} = \mathbf{0}$ .

Looking back to (1.21b), we can write the linearized least-squares term (*i.e.*, negative-log factor) for this measurement as

$$\left\| \left( \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \right) \right) - \mathbf{H}_i \boldsymbol{\xi} \right\|_{\boldsymbol{\Sigma}_i}^2, \quad (2.18)$$

where the only unknown is our pose perturbation,  $\boldsymbol{\xi}$ . After combining this with other factors and then solving for the optimal updates to our state variables, including  $\boldsymbol{\xi}^*$ , we need to update our initial guess,  $\mathbf{T}_w^{c^0}$ . For this, we must return to the perturbation scheme we chose in (2.10) and update according to

$$\mathbf{T}_w^{c^0} \leftarrow \mathbf{T}_w^{c^0} \oplus \boldsymbol{\xi}^*, \quad (2.19)$$

to ensure our solution,  $\mathbf{T}_w^{c^0}$ , remains in SE(3). As usual, optimization proceeds iteratively until the change in all the state variable updates (including  $\boldsymbol{\xi}$ ) is sufficiently small. Note that (2.19) applies when using pose perturbations on the right-hand side as done in (2.10), but this was a choice and in some cases perturbing on the left may be preferable – see Barfoot [47] for more details on this issue.

Often we will encounter measurements that also live on manifolds. For example, we might receive a noisy pose measurement,  $\mathbf{Z}_i$ , of  $\mathbf{T}_w^c \in \text{SE}(3)$ . In this case, we formulate the error in the Lie algebra so that

$$\left\| \text{Log}(\mathbf{T}_w^c \mathbf{Z}_i^{-1}) \right\|_{\Sigma_i}^2 \approx \left\| \text{Log}(\mathbf{T}_w^{c^0} \mathbf{Z}_i^{-1}) - \mathbf{H}_i \boldsymbol{\xi} \right\|_{\Sigma_i}^2, \quad (2.20)$$

where  $\mathbf{T}_w^c$  is perturbed as before and  $\mathbf{H}_i$  is the appropriate Jacobian for this error [47]. In this way, measurements of any type can be handled in the factor-graph paradigm.

To recap, we have shown how to carry out unconstrained optimization for a state variable that is a member of a Lie group. Although our example was specific to a three-dimensional pose variable, other Lie groups can be optimized in a similar manner. The key is to arrive at a situation as in (2.16) where the measurement function has been linearized with respect to a perturbation in the *Lie algebra*. Most times, as in our example, this can be done analytically. However, it is also straightforward to compute the required Jacobian,  $\mathbf{H}_i$ , numerically or through automatic differentiation (by exploiting the chain rule and some primitives for Lie groups). Stepping back a bit, this approach to optimizing a function of a Lie group member is an example of *Riemannian optimization* [106]. By exploiting the Lie algebra, which is also the *tangent space* of a manifold, we constrain the optimization to be *tangent* to the manifold of poses (or rotations). By carrying out the update according to (2.19), we are *retracting* our update back onto the manifold. Riemannian optimization is a very general concept that can be applied to quantities that live on manifolds that are not matrix Lie groups as well. Retractions other than the matrix exponential are also possible within the manifold-optimization framework (*e.g.*, see Dellaert et al. [255] or Barfoot et al. [49]).

*Riemannian optimization*

#### 2.1.4 Uncertainty and Lie Groups

We often represent uncertainty in our estimates by considering that the state vector is a random variable that follows some distribution. Gaussian distributions are the

most common as discussed in Section 1.2.2. For a vector variable,  $\mathbf{x}$ , we can write

$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (2.21)$$

where  $\boldsymbol{\mu}$  is the mean and  $\boldsymbol{\Sigma}$  is the covariance matrix. Notably, we have broken out the state into the *sum* of the mean and zero-mean noise,  $\boldsymbol{\delta}$ .

For Lie groups, we need to redefine how noise is combined with the state since simply adding noise to, for example, a rotation matrix  $\mathbf{R} \in \text{SO}(d)$ , will break the group closure property (*i.e.*, the result will no longer be a valid rotation matrix). Instead, we typically use the surjective-only mapping of the matrix exponential to combine noise,  $\boldsymbol{\delta}$ , with a ‘mean’ quantity,  $\bar{\mathbf{R}} \in \text{SO}(d)$ , as follows:

$$\mathbf{R} = \bar{\mathbf{R}} \oplus \boldsymbol{\delta} = \bar{\mathbf{R}} \text{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (2.22)$$

where it is now guaranteed that  $\mathbf{R} \in \text{SO}(d)$ . A similar approach can be followed for  $\text{SE}(d)$ :

$$\mathbf{T} = \bar{\mathbf{T}} \oplus \boldsymbol{\delta} = \bar{\mathbf{T}} \text{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (2.23)$$

where it is now guaranteed that  $\mathbf{R} \in \text{SE}(d)$  and naturally  $\boldsymbol{\delta}$  and  $\boldsymbol{\Sigma}$  must be appropriately sized. To learn more see, for example, Long et al. [686], Barfoot and Furgale [48]. We remark that in (2.22) we essentially induced a noise distribution over rotations by defining a Gaussian distribution in the tangent space and mapping it to a rotation using the exponential map. However, it is also possible to directly define distributions over rotations, and in some cases this leads to computational advantages, as we will observe in Chapter 6.

### 2.1.5 Lie Group Extras

There is a lot more that we could say about Lie groups [1030, 106] but have so far restrained ourselves in the interest of keeping things simple. We use this section to collect a few more useful facts that come up later in this and other chapters. Further details of carrying out derivatives of functions of Lie group elements will be provided in Section 4.3.

#### 2.1.5.1 The $\oplus$ and $\ominus$ Operators

We have already seen the use of the  $\oplus$  operator to compose a Lie algebra vector with a Lie group member. For  $\text{SE}(d)$  we have

$$\mathbf{T} = \mathbf{T}^0 \oplus \boldsymbol{\xi} = \mathbf{T}^0 \text{Exp}(\boldsymbol{\xi}) = \mathbf{T}^0 \exp(\boldsymbol{\xi}^\wedge) \in \text{SE}(d). \quad (2.24)$$

We often have occasion to consider the ‘difference’ of two Lie group elements and for this we can also define the  $\ominus$  operator. Again for  $\text{SE}(d)$  we have

$$\mathbf{T} = \mathbf{T}^0 \text{Exp}(\boldsymbol{\xi}) \quad \rightarrow \quad \text{Exp}(\boldsymbol{\xi}) = \mathbf{T}^{0^{-1}} \mathbf{T} \xrightarrow{\text{denoted as:}} \boldsymbol{\xi} = \mathbf{T} \ominus \mathbf{T}^0, \quad (2.25a)$$

$$\boldsymbol{\xi} = \mathbf{T} \ominus \mathbf{T}^0 = \text{Log}(\mathbf{T}^{0^{-1}} \mathbf{T}) = \log(\mathbf{T}^{0^{-1}} \mathbf{T})^\vee \in \text{se}(d). \quad (2.25b)$$

These operators are a nice way to abstract away the details of these operations.

### 2.1.5.2 Inverses

Sometimes when we are carrying out perturbations, we have need to perturb the inverse of a rotation or transformation matrix. In the  $\text{SE}(d)$  case, we simply have that

$$(\mathbf{T}^0 \oplus \boldsymbol{\xi})^{-1} = \text{Exp}(\boldsymbol{\xi})^{-1} \mathbf{T}^{0^{-1}} = \text{Exp}(-\boldsymbol{\xi}) \mathbf{T}^{0^{-1}} = (-\boldsymbol{\xi}) \oplus \mathbf{T}^{0^{-1}}, \quad (2.26)$$

where we see the perturbation moves from the right to the left with a negative sign.

### 2.1.5.3 Adjoints

*Lie group adjoint*

The *adjoint* of a Lie group is a way of describing the elements of that group as linear transformations of its Lie algebra, which we recall is a vector space. For  $\text{SO}(d)$ , the adjoint representation is the same as the group itself, so we omit the details. For  $\text{SE}(d)$ , the adjoint differs from the group's primary representation and so we use this section to provide some details. The adjoint will prove to be an essential tool when setting up state estimation problems, particularly for  $\text{SE}(d)$ .

The *adjoint map* of  $\text{SE}(d)$  for a given pose  $\mathbf{T}$  transforms a Lie algebra element  $\boldsymbol{\xi}^\wedge \in \text{se}(d)$ , in the local frame of  $\mathbf{T}$ , to another element  $\boldsymbol{\epsilon}^\wedge \in \text{se}(d)$  in the global frame, that is,

$$\boldsymbol{\epsilon}^\wedge \oplus \mathbf{T} = \mathbf{T} \oplus \boldsymbol{\xi}^\wedge \quad (2.27)$$

according to a map known as the *inner automorphism* or *conjugation*:

$$\boldsymbol{\epsilon}^\wedge = \text{Ad}_{\mathbf{T}} \boldsymbol{\xi}^\wedge = \mathbf{T} \boldsymbol{\xi}^\wedge \mathbf{T}^{-1}. \quad (2.28)$$

We can equivalently express the output of this map using the Adjoint matrix  $\text{Ad}(\mathbf{T})$  that linearly transforms  $\boldsymbol{\xi} \in \mathbb{R}^6$  to  $\mathbb{R}^6$ , such as:

$$\text{Ad}_{\mathbf{T}} \boldsymbol{\xi}^\wedge = (\text{Ad}(\mathbf{T}) \boldsymbol{\xi})^\wedge. \quad (2.29)$$

We will refer to  $\text{Ad}(\mathbf{T})$  as the *adjoint representation* of  $\text{SE}(d)$  at  $\mathbf{T}$ .

The  $(2d) \times (2d)$  transformation matrix,  $\text{Ad}(\mathbf{T})$ , can be constructed directly from the components of the  $(d+1) \times (d+1)$  homogeneous transformation matrix:

$$\text{Ad}(\mathbf{T}) = \text{Ad} \begin{pmatrix} [\mathbf{R} & \mathbf{t}] \\ [\mathbf{0} & 1] \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t}^\wedge \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}. \quad (2.30)$$

One situation in which adjoints are useful in our estimation problems is to manipulate perturbations from one side of a known transformation to another as in

$$\mathbf{T} \text{Exp}(\boldsymbol{\xi}) = \text{Exp}(\text{Ad}(\mathbf{T}) \boldsymbol{\xi}) \mathbf{T}, \quad (2.31)$$

which we emphasize does not require approximation.

*Lie group Jacobian*

Every Lie group also has a Jacobian associated with it, which allows us to relate changes in an element of the group to elements of its algebra. For the case of  $\text{SO}(d)$ , for example, the common kinematic equation (*i.e.*, Poisson's equation) relating a rotation matrix,  $\mathbf{R}_a^b \in \text{SO}(d)$  for  $d$  either 2 or 3, to angular velocity,  $\boldsymbol{\omega}^b \in \mathbb{R}^{2d-3}$ , is

$$\dot{\mathbf{R}}_a^b = \boldsymbol{\omega}^{b\wedge} \mathbf{R}_a^b. \quad (2.32)$$

Note also that  $\boldsymbol{\omega}^b$  represents the angular velocity of  $\mathcal{F}^a$  with respect to  $\mathcal{F}^b$ , expressed in  $\mathcal{F}^b$ . If we parameterize  $\mathbf{R}_a^b = \text{Exp}(\boldsymbol{\theta})$ , then we can equivalently write

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}(\boldsymbol{\theta}) \boldsymbol{\omega}^b, \quad (2.33)$$

where  $\mathbf{J}(\boldsymbol{\theta})$  is the (left) Jacobian of  $\text{SO}(d)$ . A place where this Jacobian is quite useful is when combining expressions involving products of matrix exponentials. For example, we have that

$$\text{Exp}(\boldsymbol{\theta}_1)\text{Exp}(\boldsymbol{\theta}_2) \approx \text{Exp}(\boldsymbol{\theta}_2 + \mathbf{J}(\boldsymbol{\theta}_2)^{-1}\boldsymbol{\theta}_1), \quad (2.34)$$

where  $\boldsymbol{\theta}_1$  is assumed to be ‘small’. The series expression for  $\mathbf{J}(\boldsymbol{\theta})$  is

$$\mathbf{J}(\boldsymbol{\theta}) = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\boldsymbol{\theta}^\wedge)^n, \quad (2.35)$$

and a closed-form expression can be found in Barfoot [47]. We will overload and write  $\mathbf{J}(\boldsymbol{\xi})$  for the (left) Jacobian of  $\text{SE}(d)$  where the context should inform which is meant. Jacobians of Lie groups are at the core of differentiable neural network methods that estimate poses (see Section 4.2).

## 2.2 Continuous-Time Trajectories

*Continuous-time trajectories* offer a way to represent smooth robot motions. In our development so far, we have assumed that a discrete sequence of poses along a trajectory is to be estimated. However, robots typically move fairly smoothly through the world, which motivates the use of a smoother representation of trajectory. Continuous-time trajectories come primarily in two varieties: *parametric* methods combine known temporal basis functions into a smooth trajectory. Typically, these temporal basis functions are chosen to have *local support* (*e.g.*, piecewise polynomials / splines), which ensures the factor graph remains sparse, as we will see. *Nonparametric* methods have higher representational power by making use of *kernel functions*. Specifically, a one-dimensional *Gaussian process (GP)* with time as the independent variable can be used to represent a trajectory. When an appropriate physically motivated kernel is chosen, we will see that the factor graph associated with a GP also remains very sparse.

In addition to trajectory smoothness, the use of a continuous-time trajectory can

*continuous-time trajectory*

be particularly useful when working with high-rate (such as inertial measurement units; see Section 11.2) and/or asynchronous sensors. In the factor-graph examples that we have considered so far, we added robot poses to the factor graph for each newly collected measurement (*e.g.*, to model that the current pose is taking a landmark measurement). This quickly leads to unwieldy factor graphs when using high-rate sensors or when different sensors collect measurements at different time instants. Below, we will see that we can easily represent the trajectory with a number of variables that is much smaller than the number of measurements, to keep things tractable. This is particularly useful for *motion-distorted* sensors such as spinning lidars (see Section 8.2.2.1) and radars and even rolling-shutter cameras; using continuous-time trajectories we can account for the exact time stamp of each point or pixel and relate them to the trajectory at that instant.

Finally, after MAP inference, continuous-time trajectories allow us to efficiently query the trajectory at any time of interest, not just at the measurement times. We can both interpolate and extrapolate (with caution), which can be useful for consumers of our SLAM outputs. Separating the roles of measurements times, estimation variables, and query times, is a major advantage of both parametric and nonparametric continuous-time methods.

### 2.2.1 Splines

The idea with parametric continuous-time trajectory methods is to write the pose as a weighted sum of  $K$  known *temporal basis functions*,  $\Psi_k(t)$ :

$$\mathbf{p}(t) = \sum_{k=1}^K \Psi_k(t) \mathbf{c}_k, \quad (2.36)$$

where the  $\mathbf{c}_k$  are the unknown *coefficients*. For now, we return to a vector-space explanation and discuss implementation on Lie groups in a later section. The basis functions are typically chosen to be *splines*, which are piecewise polynomials (*e.g.*, B-splines, cubic Hermite polynomials); splines are advantageous because they have *local support* meaning outside of their local region of influence they go to zero. The setup is depicted in Figure 2.3. In this example, at each instant of time only four basis functions are nonzero, which we see results in a sparse factor graph.

The main difference, as compared to our earlier discrete-time development, is that we have coefficient variables instead of pose variables, but this is completely compatible with the general factor-graph approach. Now, when we observe a landmark,  $\ell$ , at a particular time,  $t_i$ , the sensor model is

$$\mathbf{z}_i = \mathbf{h}_i(\mathbf{p}(t_i), \ell) + \boldsymbol{\eta}_i. \quad (2.37)$$

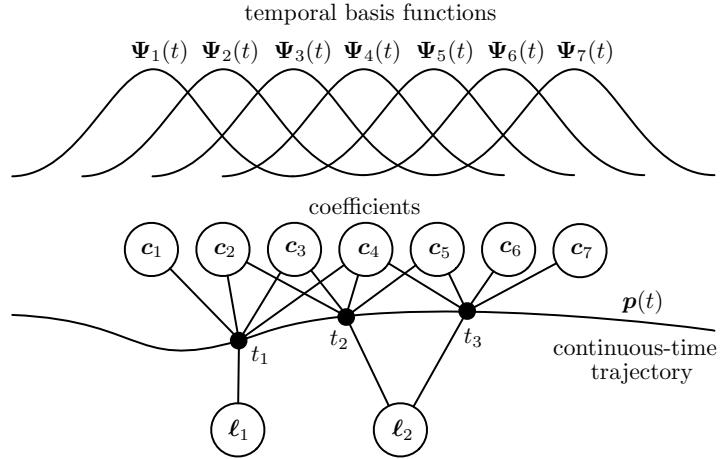


Figure 2.3 A parametric spline can be used to represent a continuous-time trajectory. In this example, the pose at a given time  $p(t)$  is assembled as a weighted sum of known temporal basis functions  $\Psi_k(t)$  with local support; at most four basis functions are nonzero at a given time. This results in each landmark measurement being represented by a *quinary* (five-way) factor between four coefficient variables and one landmark variable. The overall factor graph is still very sparse.

Inserting (2.36) we have

$$\mathbf{z}_i = \mathbf{h}_i \left( \sum_{k=1}^K \Psi_k(t_i) \mathbf{c}_k, \boldsymbol{\ell} \right) + \boldsymbol{\eta}_i. \quad (2.38)$$

As mentioned above, if our basis functions are chosen to have local support, then only a small subset of the coefficients will be active at  $t_i$ . If we let  $\mathbf{x}_i = [\mathbf{c}_i^\top \ \boldsymbol{\ell}^\top]^\top$  represent the active coefficient variables at  $t_i$  as well as the landmark variable, then we are back to being able to write the measurement function as

$$\mathbf{z}_i = \mathbf{h}_i(\mathbf{x}_i) + \boldsymbol{\eta}_i, \quad (2.39)$$

whereupon we can use our general approach to construct the nonlinear least-squares problem and optimize.

Moreover, if our basis functions are sufficiently differentiable, we can easily take the derivative of our pose trajectory,

$$\dot{\mathbf{p}}(t) = \sum_{k=1}^K \dot{\Psi}_k(t) \mathbf{c}_k \quad (2.40)$$

so that we can handle sensor outputs that are functions of, say, velocity or even higher derivatives while still optimizing the same coefficient variables. We simply need to compute the derivatives of our basis functions,  $\dot{\Psi}_k(t)$ .

Finally, once we have solved for the optimal coefficients through MAP inference, we can then query the trajectory (or its derivatives) at *any* time of interest using (2.36) or (2.40). If we compute the covariance of the estimated coefficients during inference (*e.g.*, by inverting the information matrix), this can also be mapped through to covariance of a queried pose (or derivative) quite easily since (2.36) or (2.40) are linear relationships; and, local support in the basis functions implies only the appropriate marginal covariance is needed from the coefficients.

### 2.2.2 From Parametric to Nonparametric

The main challenge with basic parametric continuous-time methods is that we must decide what type and how many basis functions to use. If we have too many basis functions, it becomes very easy to overfit to the measurement data. If we have too few basis functions, we may not have sufficient capacity to represent the true shape of the trajectory, resulting in an overly smooth solution. This challenge is partly addressed by moving to a nonparametric method.

To simplify the explanation slightly, in this section we will assume for now that there are no landmark variables, and focus only on pose variables. Using the parametric approach introduced in the previous section, our linearized least-squares term (negative-log factor) will have the form

$$\|(\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)) - \mathbf{H}_i \Psi_i \delta_{c,i}\|_{\Sigma_i}^2, \quad (2.41)$$

where  $\mathbf{x}_i^0$  is the current solution (active coefficients),  $\delta_{c,i}$  is the update (to the active coefficients),  $\Psi_i$  is the stacking of all basis functions active (and evaluated) at  $t_i$ , and the Jacobian,  $\mathbf{H}_i$ , is given by

$$\mathbf{H}_i = \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}} \Big|_{\mathbf{x}_i^0}. \quad (2.42)$$

Gathering quantities into larger matrices as before, we can write our least-squares problem as

$$\delta_c^* = \arg \min_{\delta_c} \left( \|\mathbf{b} - \mathbf{A} \Psi \delta_c\|^2 + \|\delta_c\|^2 \right), \quad (2.43)$$

where we now include a *regularizer term*,  $\|\delta_c\|^2$ , that seeks to keep the *description length* of our solution reasonable (*i.e.*, we prefer spline coefficients to be closer to zero). The regularizer term helps to avoid the over-fitting problem mentioned above. The optimal solution will be given by

$$(\Psi^\top \mathbf{A}^\top \mathbf{A} \Psi + \mathbf{I}) \delta_c^* = \Psi^\top \mathbf{A}^\top \mathbf{b}, \quad (2.44)$$

which would allow us to compute the optimal update for the coefficients,  $\delta_c^*$ . However, what we typically care about is to produce an estimate for the pose, not the spline coefficients (they are a means to an end). The optimal update to the

pose variables at the measurement times is actually  $\delta^* = \Psi\delta_c^*$ . With a little bit of algebra, we can show that

$$(\mathbf{A}^\top \mathbf{A} + \mathbf{K}^{-1}) \delta^* = \mathbf{A}^\top \mathbf{b}, \quad (2.45)$$

which is a modified version of the *normal equations*, first introduced in (1.25). The *kernel matrix*,  $\mathbf{K} = \Psi\Psi^\top$ , serves as a regularization or smoothing function. The *kernel matrix* careful reader will notice that (2.45) represents a larger linear system of equations than (2.44) because there are more poses than basis function coefficients. However, in the end we will be able to reduce the size of the linear system we need to solve in our nonparametric approach by using built-in interpolation capabilities. For now, we will work with (2.45) and come back to this issue towards the end of the section.

To move away from explicit basis functions, we can employ the so-called *kernel trick*, which replaces the explicit inner product of basis functions with evaluations of a chosen *kernel function*,  $\mathcal{K}(t, t')$  (*e.g.*, squared-exponential). We can see that in (2.45) it is only the *inner product* of the basis functions that is required to build the kernel matrix. The kernel matrix is then  $\mathbf{K} = [\mathcal{K}(t_i, t_j)]_{ij}$ , which is to say we populate it with evaluations of the kernel function at every pairing of measurement times. We can now refer to this as a *nonparametric* method since we are no longer estimating the coefficients (*i.e.*, parameters) of a spline. We do, however, have to tune the *hyperparameters* of our chosen kernel function (*e.g.*, length scale for squared exponential) to achieve the desired trajectory smoothness.

Since we need the *inverse* kernel matrix right away in (2.45), it would seem to be expensive to formulate things this way. However, the next section shows how we can choose a kernel function that guarantees that we have a very sparse inverse kernel matrix and therefore a sparse factor graph.

### 2.2.3 Gaussian Processes

We will construct a family of kernel functions that by design results in a sparse inverse kernel matrix and corresponding factor graph. We saw in the last section that we could swap out our basis functions for a kernel function, creating a non-parametric continuous-time method. However, if done naively, this could result in a dense inverse kernel matrix, which is undesirable. In this section, we come at things from a slightly different direction. As a teaser, Figure 2.4 shows an example of a factor graph resulting from the ideas in this section, which we see remains sparse yet results in smooth trajectories.

We start by choosing a linear, time-invariant, stochastic differential equation (SDE) driven by white noise:<sup>2</sup>

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{L}\mathbf{w}(t), \quad (2.46)$$

<sup>2</sup> It is also possible to include control inputs in this equation but we omit them in the interest of simplicity.

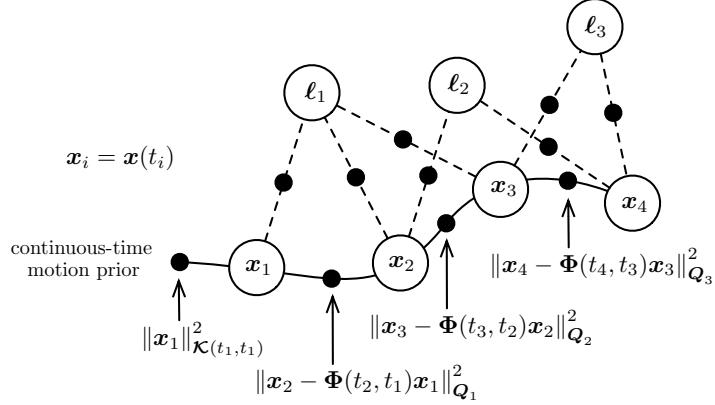


Figure 2.4 Example of Gaussian process (GP) continuous-time factor graph. The motion prior is based on a kernel function derived from a stochastic differential equation (SDE) for Markovian state  $\mathbf{x}(t)$ . This results in a very sparse set of factors: a single unary factor at the initial state and then binary factors linking consecutive states.

#### Gaussian process

where  $\mathbf{w}(t) = \mathcal{GP}(\mathbf{0}, \mathbf{Q}\delta(t - t'))$  is a zero-mean white noise Gaussian process,  $\mathbf{Q}$  is a power-spectral density matrix, and  $\delta(\cdot)$  is the Dirac delta function. The idea is that this will serve as a motion prior. We can integrate this SDE once in closed form:

$$\mathbf{x}(t) = \Phi(t, t_1)\mathbf{x}(t_1) + \int_{t_1}^t \Phi(t, s)\mathbf{L}\mathbf{w}(s) ds, \quad (2.47)$$

where  $\Phi(t, s) = \exp(\mathbf{A}(t - s))$  is known as the *transition function* and  $t_1$  is the time stamp of the first measurement. The function,  $\mathbf{x}(t)$ , is also a Gaussian process. To keep the explanation simple, if we assume the mean of the initial state is zero,  $E[\mathbf{x}(t_1)] = \mathbf{0}$ , then the mean will remain zero for all subsequent times. The covariance function of the state (*i.e.*, the kernel function),  $\mathcal{K}(t, t')$ , can be calculated as

$$\mathcal{K}(t, t') = \Phi(t, t_1)\mathcal{K}(t_1, t_1)\Phi(t', t_1)^\top + \int_{t_1}^{\min(t, t')} \Phi(t, s)\mathbf{L}\mathbf{Q}\mathbf{L}^\top\Phi(t', s)^\top ds, \quad (2.48)$$

which looks daunting. However, we can evaluate this kernel function at all pairs of measurement times (*i.e.*, build the kernel matrix) using the tidy relation

$$\mathbf{K} = \Phi\mathbf{Q}\Phi^\top, \quad (2.49)$$

where  $\mathbf{Q} = \text{diag}(\mathcal{K}(t_1, t_1), \mathbf{Q}_1, \dots, \mathbf{Q}_M)$ ,  $\mathbf{Q}_i = \int_{t_{i-1}}^{t_i} \Phi(t_i, s) \mathbf{L} \mathbf{Q} \mathbf{L}^\top \Phi(t_i, s)^\top ds$ , and

$$\Phi = \begin{bmatrix} \mathbf{I} & & & & \\ \Phi(t_2, t_1) & \mathbf{I} & & & \\ \Phi(t_3, t_1) & \Phi(t_3, t_2) & \mathbf{I} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \Phi(t_{M-1}, t_1) & \Phi(t_{M-1}, t_2) & \Phi(t_{M-1}, t_3) & \cdots & \mathbf{I} \\ \Phi(t_M, t_1) & \Phi(t_M, t_2) & \Phi(t_M, t_3) & \cdots & \Phi(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \quad (2.50)$$

with  $M$  the last measurement time index. However, since it is the *inverse* kernel matrix that we want in (2.45),  $\mathbf{K}^{-1} = \Phi^{-\top} \mathbf{Q}^{-1} \Phi^{-1}$ , we can compute this directly. The middle matrix,  $\mathbf{Q}$ , is block-diagonal and so its inverse can be computed one diagonal block at a time. Importantly, when we compute the inverse of  $\Phi$ , we find

$$\Phi^{-1} = \begin{bmatrix} \mathbf{I} & & & & \\ -\Phi(t_2, t_1) & \mathbf{I} & & & \\ & -\Phi(t_3, t_2) & \mathbf{I} & & \\ & & -\Phi(t_4, t_3) & \ddots & \\ & & & \ddots & \mathbf{I} \\ & & & & -\Phi(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \quad (2.51)$$

which is all zeros except for the main block-diagonal and one block-diagonal below. Thus, when we construct the inverse kernel matrix,  $\mathbf{K}^{-1}$ , it will be *block-tridiagonal*, for any length of trajectory. Based on our earlier discussions about factor graphs, we know that the sparsity of the left-hand side in (2.45) is closely tied to the factor-graph structure. In this case,  $\mathbf{K}^{-1}$  serves as a motion prior over the entire trajectory, but it is easily described using a very sparse factor graph. Figure 2.4 shows how the block-tridiagonal structure of  $\mathbf{K}^{-1}$  turns into a factor graph.

The reason  $\mathbf{K}^{-1}$  has such a sparse factor graph is that we started from an SDE whose state,  $\mathbf{x}(t)$ , is *Markovian*. Practically speaking, what this means is that depending on the motion prior that we want to express using (2.46), we may need to use a higher-order state, *i.e.*, not simply the pose but also some of its derivatives. For example, if we want to use the so-called ‘constant-velocity’ prior, our SDE can be chosen to be

$$\underbrace{\begin{bmatrix} \dot{\mathbf{p}}(t) \\ \dot{\mathbf{v}}(t) \end{bmatrix}}_{\dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_A \underbrace{\begin{bmatrix} \mathbf{p}(t) \\ \mathbf{v}(t) \end{bmatrix}}_{\mathbf{x}(t)} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}}_L \mathbf{w}(t), \quad (2.52)$$

where the state now comprises pose and its derivative,  $\mathbf{v}(t) = \dot{\mathbf{p}}(t)$ . Due to the use of this augmented state, this is sometimes referred to as *simultaneous trajectory estimation and mapping (STEAM)*, a variation of SLAM.

This formulation of continuous-time trajectory estimation is really an example of *Gaussian process regression* [906]. By making this connection, once we have solved

*simultaneous  
trajectory  
estimation  
and mapping*

at the measurement times, we can easily query the trajectory at other times of interest using GP interpolation (for both mean and covariance); with our sparse kernel approach, the cost of each query is constant time with respect to the number of measurements,  $M$ , as it only involves the estimated states at the two times bracketing the query.

Importantly, we can also use the resulting GP interpolation scheme to reduce the number of control points needed (*i.e.*, we do not need one at every measurement time), which is similar to the idea of *GP inducing points*. For example, we might put one control point per lidar scan but still make use of all the individual time stamps of each point gathered during a sweep. This last point is quite important because in contrast to discrete-time estimation, the measurement times, the estimation times, and the query times can now all be different in this continuous formulation. Moreover, in the GP approach we do not need to worry about overfitting by including too many estimation times as the kernel provides proper regularization. However, we still need enough estimation times to capture the detail of the trajectory.

#### 2.2.4 Spline and GPs on Lie Groups

It is also possible to use both splines and GP continuous-time methods when the state lives on a manifold. In the case that the manifolds are Lie groups, both methods make use of the Lie algebra to accomplish this, but in different ways. We begin with splines and then move to Gaussian processes.

##### 2.2.4.1 Splines on Lie Groups

*cumulative formulation  
of splines*

The key to making splines work on Lie groups is to use a *cumulative formulation*. For a vector space, we can simplify (2.36) by assuming we are using the same basis functions for all degrees of freedom so that we can write

$$\mathbf{p}(t) = \sum_{k=1}^K \psi_k(t) \mathbf{p}_k, \quad (2.53)$$

where the  $\mathbf{p}_k$  are now *control points* of our spline (replacing the earlier coefficients) and the basis functions,  $\psi_k(t)$ , are now scalar. Then, we can rewrite this in cumulative form as

$$\mathbf{p}(t) = \psi_1^c(t) \mathbf{p}_1 + \sum_{k=2}^K \psi_k^c(t) (\mathbf{p}_k - \mathbf{p}_{k-1}), \quad (2.54)$$

where

$$\psi_k^c(t) = \sum_{\ell=k}^K \psi_\ell(t) \quad (2.55)$$

are the *cumulative basis functions*.

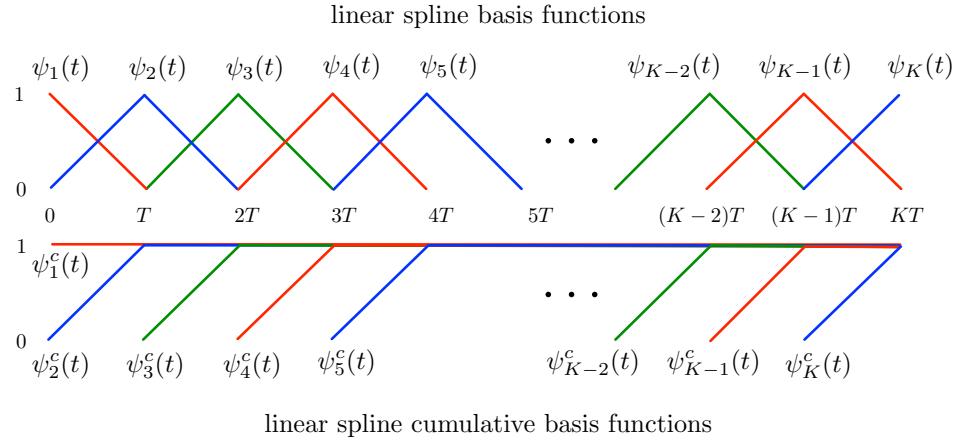


Figure 2.5 Example of linear spline basis functions both in (top) normal and (bottom) cumulative form.

For example, if we want to have *linear* interpolation with uniform temporal spacing,  $T$ , the basis functions are

$$\psi_1(t) = \begin{cases} 1 - \alpha_1(t) & 0 \leq t < T \\ 0 & \text{otherwise} \end{cases}, \quad \psi_K(t) = \begin{cases} \alpha_K(t) & (K-1)T \leq t < KT \\ 0 & \text{otherwise} \end{cases}, \quad (2.56)$$

$$k = 2 \dots K-1 : \quad \psi_k(t) = \begin{cases} \alpha_{k-1}(t) & (k-2)T \leq t < (k-1)T \\ 1 - \alpha_k(t) & (k-1)T \leq t < kT \\ 0 & \text{otherwise} \end{cases}, \quad (2.57)$$

where  $\alpha_k(t) = \frac{t-(k-1)T}{T}$ . The corresponding *cumulative* basis functions are

$$\psi_1^c(t) = 1, \quad \psi_K^c(t) = \begin{cases} 0 & \leq t < (K-1)T \\ \alpha_k(t) & (k-1)T \leq t \end{cases}, \quad (2.58)$$

$$k = 2 \dots K-1 : \quad \psi_k^c(t) = \begin{cases} 0 & t < (k-1)T \\ \alpha_k(t) & (k-1)T \leq t < kT \\ 1 & kT \leq t \end{cases}. \quad (2.59)$$

Figure 2.5 shows what these basis functions look like.

The key advantage of the cumulative basis functions is that at a given time stamp, most of the basis functions are inactive. In the case of our linear spline example, we can write

$$\mathbf{p}(t) = \mathbf{p}_{k-1} + \psi_k^c(t) (\mathbf{p}_k - \mathbf{p}_{k-1}) \quad (2.60)$$

when  $(k-1)T \leq t < kT$ . We see that only a single basis function needs to be

evaluated. With higher-order splines, we will still have only a small active set at a particular time stamp.

To apply splines on a Lie group, the idea is to then use the cumulative formulation with the Lie group operator (matrix multiplication) replacing the summation. For example, in the case of a linear spline, an element of  $\text{SE}(d)$  can be written as

$$\mathbf{T}(t) = \text{Exp}(\psi_k^c(t) \text{Log}(\mathbf{T}_k \mathbf{T}_{k-1}^{-1})) \cdot \mathbf{T}_{k-1}, \quad (2.61)$$

when  $(k-1)T \leq t < kT$ . Note, we have chosen to carry out perturbations on the left-hand side this time, but a similar formulation with right perturbations is also possible. We can now insert  $\mathbf{T}(t_i)$  into any measurement expression at some time stamp  $t_i$ , linearize it with respect to the  $\mathbf{T}_k$  control points (our estimation variables), and then use it within our MAP framework. Again, with compact-support basis functions, only a few are active at a given measurement time (one in the example of linear splines).

In a bit more detail for our linear spline example, we can rewrite (2.61) as

$$\mathbf{T}(t) = (\mathbf{T}_k \mathbf{T}_{k-1}^{-1})^{\alpha_k(t)} \mathbf{T}_{k-1}. \quad (2.62)$$

When linearizing expressions involving  $\mathbf{T}(t)$ , we can make use of the optimization approach introduced in Section 2.1.3. We perturb each of the poses<sup>3</sup> so that

$$\text{Exp}(\boldsymbol{\xi}(t)) \mathbf{T}^0(t) = \left( \text{Exp}(\boldsymbol{\xi}_k) \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \text{Exp}(-\boldsymbol{\xi}_{k-1}) \right)^{\alpha_k(t)} \text{Exp}(\boldsymbol{\xi}_{k-1}) \mathbf{T}_{k-1}^0. \quad (2.63)$$

Our goal is to relate the perturbation of the interpolated pose,  $\boldsymbol{\xi}(t)$ , to those of the control points,  $\boldsymbol{\xi}_k$  and  $\boldsymbol{\xi}_{k-1}$ . As shown by Barfoot [47], this relationship can be approximated (to first order in the perturbations) as

$$\boldsymbol{\xi}(t) \approx (\mathbf{I} - \mathbf{A}(\alpha_k(t))) \boldsymbol{\xi}_{k-1} + \mathbf{A}(\alpha_k(t)) \boldsymbol{\xi}_k, \quad (2.64)$$

where

$$\mathbf{A}(\alpha_k(t)) = \alpha_k(t) \mathbf{J} \left( \alpha_k(t) \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \right) \mathbf{J} \left( \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \right)^{-1} \quad (2.65)$$

and  $\mathbf{J}(\cdot)$  is the left Jacobian of  $\text{SE}(d)$ . We can then use (2.64) to relate changes in our pose at a measurement time to the two bracketing control-point poses in order to form linearized error terms for use in MAP estimation. For example, consider the linearized measurement model in (2.16) again, where we rearrange it as an error with slightly simpler notation for the pose and its perturbation as a function of  $t$ :

$$\mathbf{e}_i(t) \approx \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}^0(t) \tilde{\boldsymbol{\ell}}_i \right) - \mathbf{H}_i \boldsymbol{\xi}(t). \quad (2.66)$$

<sup>3</sup> In this case, we are perturbing on the left side instead of the right as shown in Section 2.1.3. The reason is that if the unknown poses represent  $\mathbf{T}_w^s(t)$  (transforms points from the ‘world’ frame to the ‘sensor’ frame), we typically apply splines in the ‘sensor’ frame and so choose the perturbations to occur there as well.

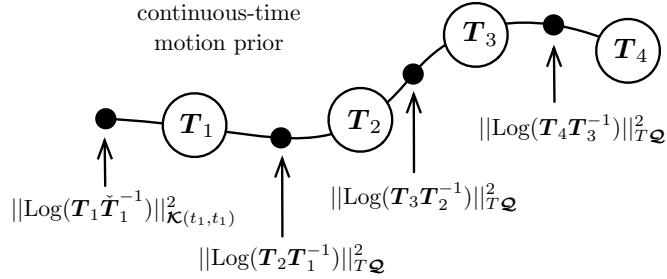


Figure 2.6 Example GP motion prior factors when using a ‘random walk’ model.

It is now a simple matter of substituting (2.64) in for  $\xi(t)$  to produce a linearized error in terms of the bracketing control points:

$$\mathbf{e}_i(t) \approx \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}(t)^0 \tilde{\ell}_i \right) - \mathbf{H}_i (\mathbf{I} - \mathbf{A}(\alpha_k(t))) \boldsymbol{\xi}_{k-1} - \mathbf{H}_i \mathbf{A}(\alpha_k(t)) \boldsymbol{\xi}_k. \quad (2.67)$$

Note, we also need to substitute  $\mathbf{T}^0(t) = \left( \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0-1} \right)^{\alpha_k(t)} \mathbf{T}_{k-1}^0$  for the nominal pose at  $t$ , both within  $\mathbf{h}_i$  and  $\mathbf{H}_i$ . We have essentially chained the derivative through our linear spline. The same process can be followed for higher-order splines as well.

#### 2.2.4.2 Gaussian Processes on Lie Groups

To use Gaussian processes on a Lie group, we will again exploit its Lie algebra to do so. Figure 2.6 provides a visual teaser of the GP motion-prior factors resulting from the ideas in this section. Note, as in the vector-space case, depending on the chosen motion prior, the control-point state may comprise additional trajectory derivatives as well.

To apply GPs on a Lie group, we will employ a local GP between a set of control-

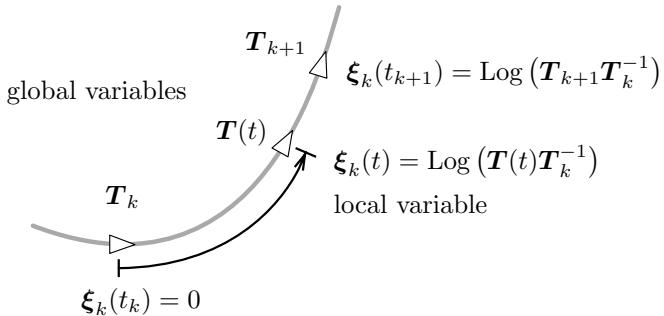


Figure 2.7 When using a GP for continuous-time estimation on Lie groups (e.g., SE( $d$ )), a local variable,  $\boldsymbol{\xi}_k(t)$ , is defined between control-point states.

point states [47], similar to splines. Figure 2.7 provides a depiction of these local variables for  $\text{SE}(d)$ . This means the SDE used to derive our kernel function operates on these local variables. For example, in the case of a ‘random-walk’ prior for  $\text{SE}(d)$ , we could choose the SDE to be

$$\dot{\xi}_k(t) = \mathbf{w}(t), \quad \mathbf{w}(t) = \mathcal{GP}(\mathbf{0}, \mathbf{Q}\delta(t - t')), \quad (2.68)$$

where we note that we have defined it using the local variable (between control points  $\mathbf{T}_k$  and  $\mathbf{T}_{k+1}$ ). The transition function for this SDE is simply  $\Phi(t, s) = \mathbf{I}$  and so stochastically integrating we have

$$\xi_k(t) = \underbrace{\xi_k(t_k)}_{\mathbf{0}} + \int_{t_k}^t \mathbf{w}(s) ds \quad (2.69)$$

and then after taking the mean and covariance we can say that the motion prior is

$$\xi_k(t) \sim \mathcal{GP}(\mathbf{0}, \min(t, t')\mathbf{Q}). \quad (2.70)$$

If we place our control-point poses uniformly spaced every  $T$  seconds then our inverse kernel matrix will be simply  $\mathbf{K}^{-1} = \Phi^{-\top} \mathbf{Q}^{-1} \Phi^{-1}$  with

$$\Phi^{-1} = \begin{bmatrix} \mathbf{I} & & \\ -\mathbf{I} & \mathbf{I} & & \\ & \ddots & \ddots & \\ & & -\mathbf{I} & \mathbf{I} \end{bmatrix}, \quad \mathbf{Q} = \text{diag}(\mathcal{K}(t_1, t_1), T\mathbf{Q}, \dots, T\mathbf{Q}). \quad (2.71)$$

The individual errors in terms of the local variables will be

$$\mathbf{e}_k = \begin{cases} \text{Log}(\mathbf{T}_1 \check{\mathbf{T}}_1^{-1}) & k = 1 \\ \mathbf{e}_{k-1}(t_k) - \xi_{k-1}(t_{k-1}) & k > 1 \end{cases}, \quad (2.72)$$

where  $\check{\mathbf{T}}_1$  is some prior initial pose value. In terms of the global variables, these same errors are

$$\mathbf{e}_k = \begin{cases} \text{Log}(\mathbf{T}_1 \check{\mathbf{T}}_1^{-1}) & k = 1 \\ \text{Log}(\mathbf{T}_k \mathbf{T}_{k-1}^{-1}) & k > 1 \end{cases}. \quad (2.73)$$

Figure 2.6 shows what the ‘random walk’ GP motion prior looks like as a factor graph. Similarly to the previous section discussing linear splines, if we want to query the trajectory at other times of interest, we can do this using GP interpolation. For the ‘random walk’ prior, this results again in linear interpolation [47]:

$$\mathbf{T}(t) = (\mathbf{T}_k \mathbf{T}_{k-1}^{-1})^{\alpha_k(t)} \mathbf{T}_{k-1}, \quad (2.74)$$

where  $\alpha_k(t) = \frac{t-(k-1)T}{T}$  and  $(k-1)T \leq t < kT$ . In contrast to the spline method, this linear interpolation results indirectly from our choice of SDE at the beginning rather than an explicit choice. Choosing higher-order SDEs at the start will result in higher-order splines for interpolation.

The last part we need to understand is how to linearize our error terms for use

in MAP estimation. To do this, we again make use of the Lie group perturbation approach detailed earlier. For example, looking at the second case in (2.73) we can write

$$\begin{aligned} \mathbf{e}_k &= \text{Log} \left( \text{Exp}(\boldsymbol{\xi}_k) \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \text{Exp}(-\boldsymbol{\xi}_{k-1}) \right) \\ &\approx \text{Log} \left( \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \right) + \boldsymbol{\xi}_k - \text{Ad} \left( \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \right) \boldsymbol{\xi}_{k-1}, \end{aligned} \quad (2.75)$$

where  $\mathbf{T}_k^0$  and  $\mathbf{T}_{k-1}^0$  are current guesses,  $\boldsymbol{\xi}_k$  and  $\boldsymbol{\xi}_{k-1}$  are the to-be-solved-for perturbations, and  $\text{Ad}(\cdot)$  is the adjoint for  $\text{SE}(d)$ . This linearized form for  $\mathbf{e}_k$  can be inserted in our standard MAP estimation framework at each iteration.

Additionally, if we want to use (2.74) to reduce the number of control points in this ‘random walk’ example, we can make use of the same approach developed for linear splines detailed in (2.67), since both methods boil down to linear interpolation between  $\text{SE}(d)$  control points. Ultimately, then, the big difference between the spline and GP approaches is that the GP approach employs motion-prior terms (see Figure 2.6) to regularize the problem, while the spline approach does not.<sup>4</sup>

### 2.3 Further Readings & Recent Trends

Much has been written about carrying out estimation on manifolds and more specifically Lie groups. The seminal reference in robotics is Chirikjian [205] (see also Chirikjian and Kyatkin [207], Chirikjian [206]; these books are well grounded in theory and describe quite general tools for handling uncertainty, even globally, on manifolds. For more details on manifold optimization, Boumal [106] is an excellent reference with an eye towards robotics applications. Barfoot [47] offers more details on handling estimation on Lie groups when the uncertainty is still somewhat compact, similar to what we have discussed in this chapter. For continuous-time estimation, Talbot et al. [1064] provides a comprehensive survey of both parametric and nonparametric methods; Barfoot [47] also discusses the nonparametric methods in some detail.

Similarly to the previous chapter, many of the current trends including handling outliers, making algorithms differentiable, and using the tools presented herein to support different types of sensors will be discussed in the following chapters, so keep reading.

<sup>4</sup> Johnson et al. [525] provide a detailed comparison between spline and GP approaches and shows that motion-prior terms can also be introduced to regularize spline methods.

# 3

## Robustness to Incorrect Data Association and Outliers

Heng Yang, Josh Mangelson, Yun Chang, Jingnan Shi,  
Niko Sunderhauf, and Luca Carlone

In Chapter 1, we have seen that factor graphs are a powerful representation to model and visualize SLAM problems, and that maximum a posteriori (MAP) estimation provides a grounded and general framework to infer variables of interest (*e.g.*, robot poses and landmark positions) given a set of measurements (*e.g.*, odometry and landmark measurements). For instance, we observed that when the measurements  $\mathbf{z}_i$  are affected by additive and zero-mean Gaussian noise with covariance  $\Sigma_i$ , MAP estimation leads to a *nonlinear least-squares* optimization:

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2, \quad (3.1)$$

where  $\mathbf{x}_i$  denotes the subset of the states involved in measurement  $i$ .<sup>1</sup> In this chapter we notice that in practice many measurements  $\mathbf{z}_i$ —possibly due to incorrect data association—may have large errors, which are far from following a zero-mean Gaussian (Section 3.1); these measurements typically induce large perturbations in the estimate  $\mathbf{x}^{\text{MAP}}$  from eq. (3.1). Therefore, we discuss how to reject gross outliers in the SLAM front-end (Section 3.2) and then focus on how to increase robustness to remaining outliers in the SLAM back-end (Section 3.3). We close the chapter with a short review of recent trends and provide extra pointers to related work (Section 3.4).

### 3.1 What Causes Outliers and Why Are They a Problem?

This section argues that outliers are inevitable in most SLAM applications and that not handling them appropriately leads to grossly incorrect estimates.

#### 3.1.1 Data Association and Outliers

To understand the cause of outlier measurements, let us consider two examples.

First, consider a landmark-based SLAM problem, where we have to reconstruct

<sup>1</sup> While for simplicity eq. (3.1) assumes that measurements belong to a vector space, the algorithms in this chapter apply to arbitrary SLAM problems where variables belong to manifolds, see Chapter 2.

the trajectory of the robot and the position of external landmarks from odometry measurements and relative observations of landmarks from certain robot poses. Assuming (as we did in Chapter 1) that the landmark measurements have zero-mean Gaussian noise leads to terms in the optimization in the form  $\|\mathbf{z}_{ij} - \mathbf{h}(\mathbf{p}_i, \ell_j)\|_{\Sigma}^2$ . These terms model the fact that a given measurement  $\mathbf{z}_{ij}$  is an observation of landmark  $\ell_j$  from pose  $\mathbf{p}_i$  up to Gaussian noise, where  $\mathbf{h}(\cdot)$  is the function describing the type of relative measurement (*e.g.*, range, bearing, etc.). In practice, the measurements  $\mathbf{z}_{ij}$  are obtained by pre-processing raw sensor data in the SLAM front-end. For instance, if the robot has an onboard camera and  $\mathbf{z}_{ij}$  is a visual observation of the bearing to a landmark  $\ell_j$ , the measurement  $\mathbf{z}_{ij}$  might be extracted by performing object (or more generally, feature) detection and matching in the image, and then computing the bearing corresponding to the detected pixels. Now, the issue is that the detections are imperfect and a landmark detected as  $\ell_j$  in the image, might be actually a different landmark in reality. This causes  $\mathbf{z}_{ij}$  to largely deviate from the assumed model. The problem of associating a measurement to a certain landmark is typically referred to as the *data association problem* and is common to many other estimation problems (*e.g.*, target tracking). Therefore, incorrect data association creates outliers in the estimation problem.

As a second example, consider a pose-graph optimization problem, where we are primarily interested in estimating the trajectory of the robot (represented as a set of poses), and the measurements are either odometry measurements (which relate consecutive poses along the trajectory) or *loop closures* (which relate non-consecutive and possibly temporally distant poses). In practice, the loop closures are detected using (*e.g.*, vision-based or lidar-based) place recognition methods, which are in charge of detecting if a pair of poses  $\mathbf{p}_i$  and  $\mathbf{p}_j$  have observed the same portion of the environment. Unfortunately current place recognition methods are prone to making mistakes and detecting loop closures between poses that are *not* observing the same scene. This is partially due to limitations of current methods, but it is often due to *perceptual aliasing*, that is the situation where two similarly looking locations actually correspond to different locations (think of two classrooms in a university building, or similarly looking cubicles in an office environment). This can be again understood as a failure of data association, where we mistakenly associate the loop closure measurement to two incorrectly chosen robot poses.

Note that outliers are not only caused by incorrect data association, but can also be caused by violations of the assumptions made in the SLAM approach. For instance, the majority of SLAM approaches assume landmarks to be static, hence detections of a moving object —even when correctly associated to that object— may lead to outlier measurements with large residuals. Similarly, sensor failure and degradation, *e.g.*, a faulty wheel encoder or dust on the camera lens, might contribute to creating outliers in the measurements.

### 3.1.2 Least-Squares in the Presence of Outliers

In the presence of outliers, the estimate resulting from the least-squares formulation (3.1) can be grossly incorrect. From the theoretical standpoint, the Gaussian noise we assumed for the measurement is “light-tailed”, in that it essentially rules out the possibility of measurements with very large error. From a more practical perspective, the outliers lead to terms in the objective function where the residual error  $r_i(\mathbf{x}) := \|\mathbf{z}_{ij} - \mathbf{h}(\mathbf{p}_i, \ell_j)\|_{\Sigma}$  is very large, when evaluated near the ground truth. Since the residuals are squared in the objective of the optimization, *i.e.*, the objective is  $\sum_i r_i(\mathbf{x})^2$ , these residuals have a disproportionately large impact on the cost, and the optimization focuses on minimizing the large terms induced by the outliers rather than making good use of the remaining (inlier) measurements.

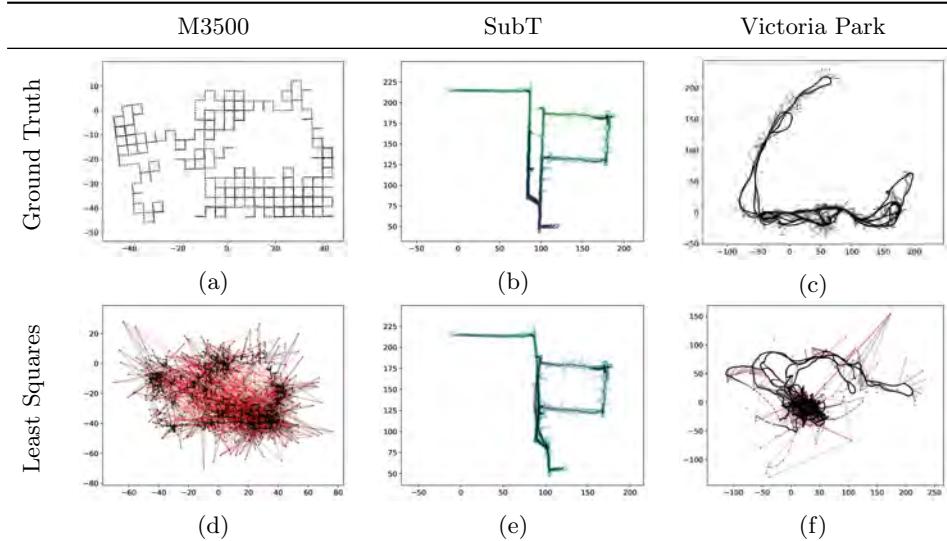


Figure 3.1 SLAM problems with outliers: (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained with the least-squares formulation in the presence of outliers. Inlier measurements are visualized as gray edges, while outliers are visualized as red edges. In the SubT dataset, we also visualize a dense map built from the SLAM pose estimate.

To illustrate this point, Figure 3.1 shows results for three SLAM problems with outliers. The first column is a simulated pose-graph optimization benchmark, known as M3500, with poses arranged in a grid-like configuration; the dataset includes 3500 2D poses and 8953 measurements. The second column is a real-world pose-graph dataset, denoted as SubT, collected in a tunnel during the DARPA Subterranean Challenge [299]; the dataset includes 682 3D poses and 3278 measurements. The third column is a real-world landmark-SLAM dataset, known as Victoria Park [806]; the dataset includes 7120 2D poses and landmarks, and 17728 measurements. Fig-

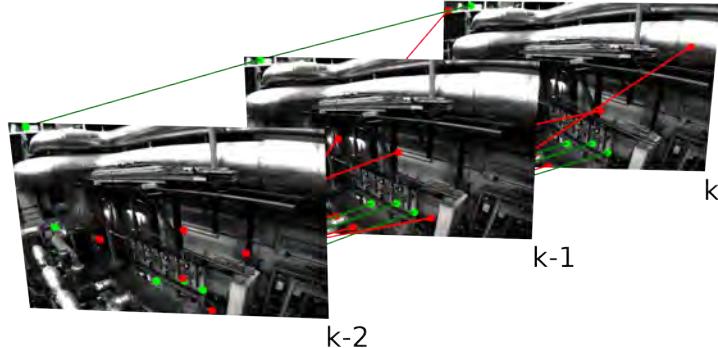


Figure 3.2 Feature tracking across three frames (collected at time  $k - 2$ ,  $k - 1$ , and  $k$ ) in a visual SLAM problem. Inliers are visualized in green: these are pixels picturing the same (static) 3D point over time. Outliers are visualized in red.

ure 3.1(a)-(c) show the ground truth trajectories for the three problems. Figure 3.1(d)-(f) show the estimate produced by the least-squares formulation in the presence of outliers. In particular, for M3500 and Victoria Park we add 15% random outliers to the (loop closures or landmark) measurements, while the SubT dataset already includes outliers. In the figure, we visualize outlier measurements in red. We observe that the presence of outliers leads to completely incorrect trajectories and map estimates. Moreover, the outliers often expose perceptual aliasing in the environment: for instance, the two similarly looking vertical corridors in the middle of the SubT dataset induce many spurious loop closures, which mislead the back-end to create a map with a single vertical corridor.

### 3.2 Detecting and Rejecting Outliers in the SLAM Front-end

The main role of the SLAM front-end is to extract intermediate representations or (pseudo-)measurements —which will be converted into factors for the back-end— from the raw sensor data. Typical SLAM front-ends accomplish this by first computing an initial set of measurements (possibly corrupted by many outliers) and then post-processing the initial set to remove outliers. This section discusses two approaches to reject outliers in the SLAM front-end: RANSAC and Pairwise Consistency Maximization.

#### 3.2.1 *R*andom *S*ample *C*onsensus (*RANSAC*)

RANSAC is a well-established tool for outlier rejection [328] and is a key component of many landmark-based SLAM systems. In order to understand what RANSAC is and its role in SLAM, consider a landmark-based visual SLAM approach.

**Example 3.1** (Outliers in landmark-based visual SLAM) A landmark-based (or feature-based) visual SLAM approach extracts 2D feature points in each image and then associates them across consecutive frames using either optical-flow-based feature tracking or descriptor-based feature matching (Figure 3.2). In particular, at time  $k$ , the approach detects 2D feature points and matches them with corresponding points observed in the previous frame (say, at time  $k - 1$ ); the matching pixels are typically referred to as *2D-2D correspondences*. Due to inaccuracies of optical flow or descriptor-based matching, this initial set of correspondences might contain outliers. Therefore, it is important to filter out gross outliers before passing them to the back-end, which estimates the robot poses and landmark positions.

RANSAC

RANSAC is a tool to quickly detect and remove outliers in the correspondences before passing them to the back-end. Detecting outliers relies on two key insights. The first insight is that in SLAM problems, inlier correspondences must satisfy geometric constraints. For instance, in our example, inlier correspondences picture the observed pixel motion of static 3D points as the camera moves. The resulting pixel motion cannot be arbitrary, but must follow a precise geometric constraint, known as the *epipolar constraint*, which dictates how corresponding pixels in two frames are related depending on the camera motion. In particular, for calibrated cameras, the epipolar constraint imposes that corresponding pixels  $\mathbf{z}_i(k - 1), \mathbf{z}_i(k)$ —picturing landmark  $i$  at time  $k - 1$  and  $k$ , respectively—satisfy

$$\mathbf{z}_i(k - 1)^T ([\mathbf{t}_k^{k-1}]_{\times} \mathbf{R}_k^{k-1}) \mathbf{z}_i(k) = 0, \quad (3.2)$$

where  $\mathbf{t}_k^{k-1}$  and  $\mathbf{R}_k^{k-1}$  are the relative position and rotation describing the (unknown) motion of the camera between time  $k - 1$  and  $k$ .<sup>2</sup> More generally, if we denote the  $i$ -th correspondence as  $\mathbf{z}_i$  (in the example above,  $\mathbf{z}_i = \{\mathbf{z}_i(k - 1), \mathbf{z}_i(k)\}$ ), these geometric constraints are in the form

$$C(\mathbf{z}_i, \mathbf{x}) \leq \gamma, \quad (3.3)$$

which states that the correspondences have to satisfy some inequality, which is possibly a function of the unknown state  $\mathbf{x}$ ; in (3.3) the parameter  $\gamma$  on the right-hand-side is typically tuned to account for the presence of noise. For instance, while ideally the epipolar constraint in (3.2) is exactly satisfied, in practice it might have small errors since the pixel detections are noisy, and hence we would relax the constraint to only require  $|\mathbf{z}_i(k - 1)^T ([\mathbf{t}_k^{k-1}]_{\times} \mathbf{R}_k^{k-1}) \mathbf{z}_i(k)| \leq \gamma$ , for some small  $\gamma$ .

The second insight is that—assuming we do not have too many outliers—we can find the inliers as the largest set of correspondences that satisfy the geometric

<sup>2</sup> Clearly, different problems will have different geometric constraints, but luckily there is a well-established literature in robotics and computer vision, that studies geometric constraints induced by different types of sensor measurements. The example in this section considers 2D-2D correspondences, and the corresponding constraints have been studied in the context of 2-view geometry in computer vision, see [437].

constraint (3.3) for some  $\mathbf{x}$ :

$$\begin{aligned} S_{CM}^* = \operatorname{argmax}_{\mathbf{x}, S \subset M} |S| \\ \text{s.t. } C(\mathbf{z}_i, \mathbf{x}) \leq \gamma, \quad \forall i \in S \end{aligned} \tag{3.4}$$

where  $M$  is the set of initial putative correspondences, and  $|S|$  denotes the cardinality (number of elements) in the subset  $S$  [725]. In words, the optimization (3.4) looks for the largest subset  $S$  of the set of putative correspondences  $M$ , such that measurements in  $S$  satisfy the geometric constraints for the same value of  $\mathbf{x}$ . Intuitively, problem (3.4) captures the intuition that the inliers (estimated by the set  $S$ ) must “agree” on the same  $\mathbf{x}$  (*e.g.*, they must all be consistent with the actual motion of the robot). Problem (3.4) is known as *consensus maximization* in computer vision. Note that (3.4) does not require solving the entire SLAM problem (which might involve many poses and landmarks), since it only involves a small portion of the SLAM state; for instance, the epipolar constraint (3.2) only involves the relative pose between two frames rather than the entire SLAM trajectory. At the same time, (3.4) is still a hard combinatorial problem, which clashes with the fast run-time requirements of typical SLAM front-ends. Therefore, rather than looking for exact solutions to (3.4), it is common to resort to quick heuristics to approximately solve (3.4).

*consensus maximization*

*RANDom SAmple Consensus* (RANSAC) is probably the most well-known approach to find an approximate solution to the consensus maximization problem in (3.4). RANSAC builds on the key assumption that  $\mathbf{x}$  in (3.4) is relatively low-dimensional and can be estimated from a small set of measurements (the so-called *minimal set*), using fast estimators (the so called *minimal solvers*).<sup>3</sup> For instance, in our visual SLAM example, one can estimate the relative motion between two camera frames using only 5 pixel correspondences, using Nister’s 5-point method [807]. Then the key idea behind RANSAC is that, instead of exhaustively checking every possible subset  $S \subset M$ , one can *sample* minimal sets of measurements looking for inliers. More in detail, RANSAC iterates the following three steps:

- 1 Sample a subset of  $n$  correspondences, where  $n$  is the size of the minimal set for the problem at hand;<sup>4</sup>
- 2 Compute an estimate  $\hat{\mathbf{x}}$  from the  $n$  sampled correspondences using a minimal solver;<sup>5</sup>
- 3 Select the correspondences  $S \subset M$  that satisfy the geometric constraint  $C(\mathbf{z}_i, \hat{\mathbf{x}}) \leq$

<sup>3</sup> The development of minimal solvers can be considered a sub-area of computer vision research, hence for typical problems it is well-understood what is the size of the minimal set and there are well-developed (and typically off-the-shelf) minimal solvers one can use.

<sup>4</sup> In our example with pixel correspondences between calibrated camera images, the minimal set has size  $n = 5$ , since 5 non-collinear measurements are sufficient to determine the pose between two cameras up to scale.

<sup>5</sup> In our example, this involves computing the relative motion (up to scale) between time  $k - 1$  and  $k$  using, *e.g.*, Nister’s 5-point method [807].

$\gamma$  for the  $\hat{\mathbf{x}}$  computed at the previous step. Store the set  $S$  if it is larger than the set computed at the previous iterations.

The set  $S$  computed in the last step is called the *consensus set* and RANSAC typically stops after computing a sufficiently large consensus set (as specified by a user parameter) or after a maximum number of iterations. RANSAC essentially attempts to sample  $n$  inliers from the set of measurements, since these are likely to “agree” with all the other inliers and hence have a large consensus set.

RANSAC is the go-to solution for many outlier-rejection problems. In particular, it quickly converges to good estimates (*i.e.*, good sets of correspondences) in problems with small number of outliers and small minimal sets. Assuming that the probability of sampling an inlier from the set of measurements is  $\omega$ ,<sup>6</sup> it is easy to conclude that the expected number of iterations RANSAC requires for finding a set of inliers is  $\frac{1}{\omega^n}$ . For instance, when  $n = 5$  and  $\omega = 0.7$  (*i.e.*, 70% of the measurements are inliers), the expected number of iterations is less than 10. This, combined with the fact that non-minimal solvers are extremely fast in practice (often allowing hundreds of iterations in a handful of milliseconds), makes RANSAC extremely appealing. Moreover, RANSAC also provides an estimate  $\hat{\mathbf{x}}$  (*e.g.*, the robot odometry), that can be useful as an initial guess for the back-end.

On the downside, RANSAC may not be the right approach for all problems. In particular, the expected number of iterations becomes impractically large when the number of inliers is small or when the minimal set is large; for instance, when  $n = 10$  and  $\omega = 0.1$ , the expected number of iterations to find a set of inliers becomes  $10^{10}$ , and terminating RANSAC after a smaller number of iterations is likely to return incorrect solutions (*i.e.*, incorrect  $\hat{\mathbf{x}}$  and correspondences). As we discuss in the next section, in the context of many SLAM problems, the assumptions of having many inliers and small minimal sets are not always valid.

### 3.2.2 Graph-theoretic Outlier Rejection and Pairwise Consistency Maximization

As we mentioned, RANSAC is very effective when the number of outliers is reasonable (say, below 70%) and the size of the minimal set is small (say, less than 8). However, environments with severe perceptual aliasing might have very high number of outliers. Moreover, not all the problems we are interested in have a fast minimal solver with a small minimal set. For instance, if we consider a pose-graph SLAM problem with  $N$  nodes, the minimal set must include at least  $N - 1$  measurements (forming a spanning tree of the pose-graph), and  $N$  is typically in the thousands.

For these reasons, this section introduces an alternative approach, known as

<sup>6</sup> Assuming that samples are drawn uniformly at random,  $\omega$  can be thought of as the fraction of measurements that are inliers.

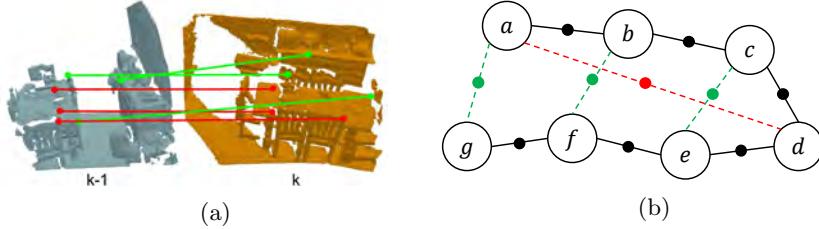


Figure 3.3 (a) 3D-3D correspondences from two RGB-D scans representing two partial views of a scene. The green lines indicate inlier correspondences and the red lines indicate outlier correspondences. (b) Pose graph with outliers in the loop closures. The dashed green lines indicate inlier loop closures while the dotted red line is an outlier loop closure.

*Pairwise Consistency Maximization (PCM)*, that, rather than sampling minimal sets, seeks to find the largest set of measurements that are internally “consistent” with one another, using graph theory. This approach can be used to sort through sets of measurements with upwards of 90% outliers and prune gross outliers before passing them to the back-end. The approach was initially proposed in [725] and extended beyond pairwise consistency in [1002, 332].

The key insight behind PCM is that for many problems one can define *consistency functions* that capture whether a pair of measurements are consistent with each other. Let’s elucidate on this point with two examples.

**Example 3.2** (Consistency Function in landmark-based visual SLAM with RGB-D cameras) A landmark-based visual SLAM approach with RGB-D cameras extracts 3D feature points in each RGB-D frame and then associates them across consecutive frames (Figure 3.3(a)). In particular, at time  $k$ , the approach detects 3D feature points and matches them with corresponding points observed in the previous frame (say, at time  $k-1$ ); the matching 3D points are typically referred to as *3D-3D correspondences*. We observe that the 3D points collected at time  $k$  and  $k-1$  ideally correspond to the same set of 3D static points observed from two different viewpoints; therefore, the distance between a pair of corresponding points  $\{\mathbf{z}_i(k-1), \mathbf{z}_j(k-1)\}$  and  $\{\mathbf{z}_i(k), \mathbf{z}_j(k)\}$  has to be constant over time (up to noise):

$$\|\mathbf{z}_i(k-1) - \mathbf{z}_j(k-1)\| - \|\mathbf{z}_i(k) - \mathbf{z}_j(k)\| \leq \gamma \quad (3.5)$$

We observe that contrary to the geometric constraints used in RANSAC, the consistency function (3.5) (i) does not depend on the state, hence it can be evaluated directly without the need for a minimal solver, and (ii) involves a pair of correspondences regardless of the size of the minimal set. While the previous example could also be solved with RANSAC,<sup>7</sup> let us now consider a higher dimensional problem.

<sup>7</sup> Motion estimation from 3D-3D correspondences admits a fast 3-point minimal solver, *e.g.*, Horn’s method [477].

**Example 3.3** (Consistency Function in pose-graph SLAM) Consider a pose-graph SLAM problem where loop closures might contain outliers due to place recognition failure and perceptual aliasing; we assume the odometry is reliable and outlier free. In order to understand if two loop closures are consistent with each other, we observe that in the noiseless case, pose measurements along cycles in the graph must compose to the identity (Figure 3.3(b)).<sup>8</sup> Therefore, a pair of loop closures  $\mathbf{T}_b^a$  (between poses  $a$  and  $b$ ) and  $\mathbf{T}_d^c$  (between poses  $c$  and  $d$ ) must satisfy:

$$\text{dist}(\mathbf{T}_b^a \cdot \bar{\mathbf{T}}_c^b \cdot \mathbf{T}_d^c \cdot \bar{\mathbf{T}}_a^d, \mathbf{I}) \leq \gamma \quad (3.6)$$

where  $\bar{\mathbf{T}}_c^b$  and  $\bar{\mathbf{T}}_a^d$  are the chain of odometry measurements from node  $b$  to node  $c$ , and from node  $d$  to node  $a$ , respectively, and  $\text{dist}$  is a suitable distance function that measures how far is  $\mathbf{T}_b^a \cdot \bar{\mathbf{T}}_c^b \cdot \mathbf{T}_d^c \cdot \bar{\mathbf{T}}_a^d$  from the identity pose. As usual,  $\gamma$  is a parameter chosen to account for the noise: measurements along a loop might not compose to the identity due to noise in the odometry and loop closures.<sup>9</sup>

More generally, a *consistency function* is a function relating two measurements and that have to satisfy a given constraint. For a pair of measurements  $\mathbf{z}_i$  and  $\mathbf{z}_j$ , the resulting *pairwise consistency constraints* are in the form:

$$F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma, \quad (3.7)$$

where  $F$  is the consistency function, and  $\gamma$  is a user-specified parameter that accounts for measurement noise. We remark that the pairwise consistency constraint are state independent, hence they can be efficiently checked without resorting to a minimal solver by just inspecting every pair of measurements.

Using (3.7), we can formulate an alternative approach for outlier rejection, which selects the largest set of measurements that are pairwise consistent:

$$\begin{aligned} \mathcal{S}_{\text{PCM}}^* &= \underset{\mathcal{S} \subset \mathcal{M}}{\operatorname{argmax}} |\mathcal{S}| \\ \text{s.t. } F(\mathbf{z}_i, \mathbf{z}_j) &\leq \gamma, \quad \forall i, j \in \mathcal{S} \end{aligned} \quad (3.8)$$

*Pairwise Consistency Maximization*

Problem (3.8) looks for the largest subset  $\mathcal{S}$  of measurements such that every pair of measurements in  $\mathcal{S}$  are pairwise consistent. We refer to this as the *pairwise consistency maximization* (PCM) problem. This problem is still combinatorial in nature, but appears slightly easier than (3.4): the problem does not involve  $\mathbf{x}$ , and the constraints  $F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma$  can be pre-computed for every pair  $(i, j)$  in  $\mathcal{M}$ . Furthermore, the problem admits a graph-theoretic interpretation, which allows solving (3.4) using well-established tools from graph theory, namely, *maximum clique* algorithms.

In order to draw a connection between problem (3.8) and graph theory, let us visualize the outlier-rejection problem as a graph  $\mathcal{G}$ , where the nodes of the graph are the putative measurements  $i \in \mathcal{M}$  and an edge exists between two nodes  $i$  and  $j$  if

<sup>8</sup> Intuitively, if we walk back along a loop in the environment we come back to our initial location.

<sup>9</sup> In practice, one would select  $\gamma$  to account for the size of the loop: intuitively, longer loops will accumulate more noise, see [725, 298].

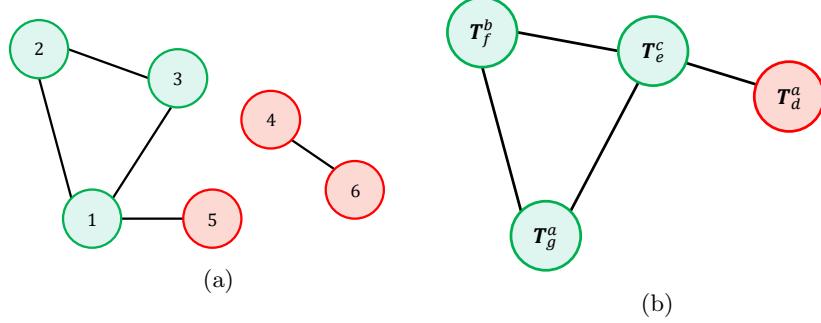


Figure 3.4 (a) Consistency graph of the 3D-3D correspondences example in Figure 3.3(a). (b) Consistency graph of the loop closures for the pose-graph example in Figure 3.3(b).

$F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma$ . This is typically called the *consistency graph*. Now problem (3.8) asks to select the largest subset of nodes  $\mathbf{S}$  such that every pair of nodes in  $\mathbf{S}$  is connected by an edge: this is exactly the definition of *maximum clique* of a graph. More in detail, a *clique* in graph theory is a subset of nodes in which every pair of nodes has an edge between them, and the *maximum clique* is the largest such subset of nodes in the graph. Therefore, the solution to problem (3.8) is the maximum clique of the consistency graph  $\mathcal{G}$ . This graph theoretic connection is really useful in practice, since the problem of finding the maximum clique for a given graph is a well-studied problem in graph theory and is called the maximum clique problem. The maximum clique problem is an NP-hard problem [1194] and hard to approximate [1311, 322], meaning that finding a solution arbitrarily close to the true solution is also NP-hard. However, dozens of potential maximum clique algorithms have been proposed, some of which can handle significantly sized problems depending on the density of the graph. The majority of proposed methods can be classified as either exact or heuristic-based methods. All of the exact algorithms are exponential in complexity and are usually based on branch and bound, while the heuristic algorithms often try to exploit some type of structure in the problem, making them faster, at the expense of not necessarily guaranteeing the optimal solution [1194]. Relatively recent works, e.g., [850], propose maximum clique algorithms that are parallelizable and able to quickly find maximum cliques in large sparse graphs.

In summary, solving the PCM problem using a maximum clique algorithm involves the following steps:

- 1 Select a Consistency Function  $F$  for the problem at hand.
- 2 Evaluate the Consistency Function for every pair of putative measurements  $(i, j) \in M$ , and create a consistency graph with edges between  $i$  and  $j$  when  $F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma$ .
- 3 Solve for the Maximum Clique of the Consistency Graph using exact or approximate maximum clique algorithms.
- 4 Return measurements  $\mathbf{S}$  in the (possibly approximate) maximum clique.

We remark that the choice of consistency function is problem-dependent. Moreover, choosing a good consistency function might largely influence the quality of the outlier rejection. For instance, one could select a dummy function  $F(\mathbf{z}_i, \mathbf{z}_j) = 0$  which always returns zero regardless of the arguments; such a function would not allow rejecting any outliers, hence making PCM ineffective. On the other hand, if we make the function such that only the inliers can pass the test, then we would exactly reject all the outliers. A selection of potential consistency functions for a broad variety of geometric problems is discussed in [1002, 332].

Before concluding this section a few remarks are in order. While we observed that PCM has the ability to handle a large number of outliers compared to RANSAC and is more suitable for certain problems (*e.g.*, pose-graph SLAM), the trade-offs between PCM and RANSAC are more nuanced. RANSAC evaluates the consistency of individual measurements using an estimate computed by a minimal solver; PCM, on the other hand, evaluates the consistency of a set of measurements to each other in a pairwise manner. In certain cases, RANSAC’s individual consistency is insufficient to evaluate the set of measurements as a whole: this is often the case in pose-graph optimization where individual consistency of a pair of loop closure measurements does not necessarily ensure pairwise consistency of the two loop closures.<sup>10</sup> On the other hand, for certain problems such as 3D-3D pose estimation (Example 3.2), the pairwise consistency function (3.5) used in PCM might be more permissive than RANSAC and lead to classifying certain outliers as inliers.

In the context of PCM, it is also important to note that exact maximum clique solvers tend to be slow in dense consistency graphs (*i.e.*, when many pairs of measurements are consistent), hence heuristic-based maximum-clique solutions may be a better choice for certain problems. Finally, for certain problems, it might be hard to design a suitable consistency function; for instance, for 2D-2D correspondences, there is no easy way to rigorously design a general consistency function due to the lack of suitable invariances (see discussion in [1002]).

### 3.3 Increasing Robustness to Outliers in the SLAM Back-end

Front-end outlier rejection, including both RANSAC and PCM, might still miss outliers and pass an outlier-contaminated set of measurements to the back-end.<sup>11</sup> As we have seen in Section 3.1.2, a handful of outliers can lead to completely wrong results when using standard least squares estimation. Therefore, it is important to enhance the back-end to be robust to remaining outliers.

In Section 3.1.2, we observed that the use of squared residuals “amplifies” the

<sup>10</sup> This is especially pronounced in the context of the multi-robot pose-graph optimization — where the goal is to estimate the trajectory of two (or more) robots jointly within a single pose-graph. In these contexts in particular, PCM has been shown to dramatically outperform RANSAC [725].

<sup>11</sup> Intuitively, both the geometric constraints (3.3) —even when evaluated at the ground truth  $\mathbf{x}$ — and the pairwise consistency constraints (3.7) are necessary (but not sufficient) conditions for measurements to be inliers. Moreover, consensus maximization and PCM are often solved with approximation algorithms that do not guarantee an optimal selection of the inliers.

impact of outlying measurements on the cost function. In this section we slightly modify the objective function in the SLAM optimization to regain robustness to outliers, following the standard theory of M-Estimation in robust statistics [495].

M-Estimation (“Maximum-likelihood-type Estimation”) is a framework for robust estimation and suggests replacing the squared loss in eq. (3.1) with a suitably chosen *robust loss* function  $\rho$ :

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \mathbf{r}_i(\mathbf{x})^2 \implies \mathbf{x}^{\text{MEST}} = \arg \min_{\mathbf{x}} \sum_i \rho(\mathbf{r}_i(\mathbf{x})). \quad (3.9)$$

The key requirement for the robust loss  $\rho$  is to be a non-negative function and grow less than quadratically for large residuals; in other words, robust loss functions need to have derivative  $\frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i} \ll \frac{\partial \|\mathbf{r}_i\|^2}{\partial \mathbf{r}_i} = 2\mathbf{r}_i$  as  $\mathbf{r}_i$  becomes large; in many cases, it is desirable for  $\frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i}$  to approach zero as  $\mathbf{r}_i$  becomes large. To elucidate this requirement, consider the case where we solve  $\min_{\mathbf{x}} \sum_i \rho(\mathbf{r}_i(\mathbf{x}))$  using gradient descent. Using the chain rule, the gradient of the objective  $f(\mathbf{x}) \doteq \sum_i \rho(\mathbf{r}_i(\mathbf{x}))$  becomes:

$$\frac{\partial f}{\partial \mathbf{x}} = \sum_i \frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i} \cdot \frac{\partial \mathbf{r}_i(\mathbf{x})}{\partial \mathbf{x}} \quad (3.10)$$

From (3.10), it is clear that if we start for a good initial guess (*i.e.*, relatively close to the ground truth), outlier measurements will have large residual and hence very small  $\frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i}$ , thus having a minor influence on the overall descent direction. Hence they will have almost no influence in the estimation. Indeed, the function  $\psi(\mathbf{r}_i) := \frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i}$  is typically referred to as the *influence function* [85].

Rather than a single choice of robust loss function, the robust estimation literature provides a “menu” of potential choices. Figure 3.5 lists common choices of loss functions. This list includes common robust losses, such as Huber, Geman-McClure, Tukey’s biweight and the truncated quadratic loss, and also includes a more radical choice, named *maximum consensus* loss. The latter is not typically listed among the loss functions in the robust estimation literature, but we mention it here, since it connects back to the consensus maximization problem we discussed in (3.4).<sup>12</sup> The choice of robust loss is fairly problem-dependent [553, 1072]. For instance, loss functions with hard cut-offs (*e.g.*, the truncated quadratic loss, where there is a sudden transition between the quadratic and the “flat” portion of the function) are preferable when a reasonable threshold for the cut-off (*i.e.*, the maximum error expected from the inliers) is known. One also has to take into account computational considerations. For instance, Huber is often used in BA problems since it is a convex function and is better-behaved during the optimization, despite leaving non-zero influence for the outliers (the influence becomes zero only if the

*M-Estimation*

*robust losses*

<sup>12</sup> Intuitively, the maximum consensus loss “counts” the outliers in the estimation problem, since it is constant (typically equal to 1) for large residuals and zero for small residuals. Therefore, minimizing such loss leads to an estimate that produces the least number of outliers. This is the same as solving the consensus maximization problem in (3.4).

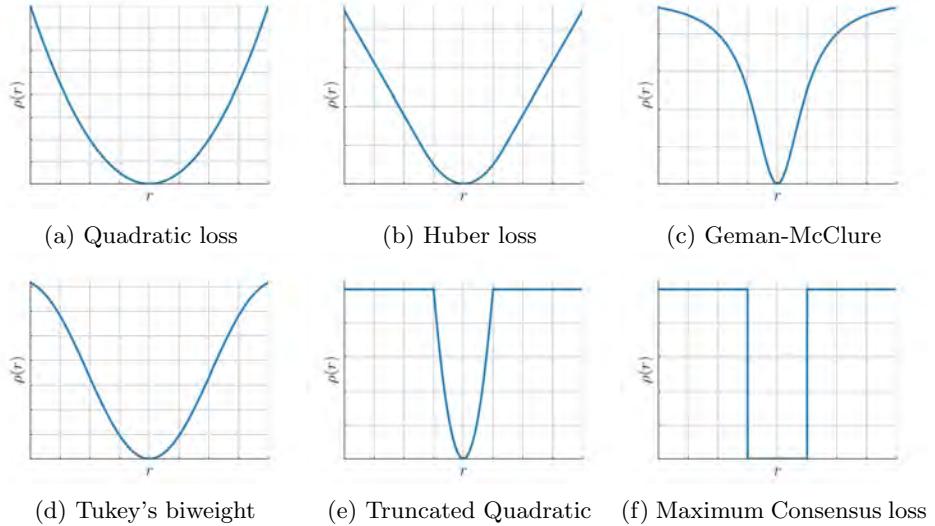


Figure 3.5 Quadratic loss and examples of robust loss functions. The shape of the robust loss functions is controlled by a parameter that controls the separation between inliers and outliers.

loss is constant for large residuals). On the other hand, the truncated quadratic and maximum consensus losses are known to be particularly insensitive to outliers, but they often require ad-hoc solvers.<sup>13</sup>

Figure 3.6(g)-(l) show the SLAM trajectories obtained by applying gradient descent to two of the robust losses mentioned above: the Huber loss and the truncated quadratic loss. Here we consider the same datasets used in Figure 3.1. We implemented the gradient descent solver using GTSAM’s NonlinearConjugateGradientOptimizer [253] with the gradientDescent flag enabled, and using robust noise models to instantiate the robust loss functions. We set the maximum number of iterations and the stopping conditions thresholds (relative and absolute tolerance) to 10000 and  $10^{-7}$ , respectively. All other parameters were left to the default GTSAM values. In the figure, an edge is colored in gray if it is correctly classified as inlier or outlier by the optimization (*i.e.*, an inlier that falls in the quadratic region of the Huber or truncated quadratic loss); it is colored in red if it is an outlier incorrectly classified as an inlier (a “false positive”); it is colored in blue if it is an inlier incorrectly classified as an outlier (a “false negative”). Compared to (non-robust) least squares optimization (Figure 3.1(d)-(f)), we note that the use of the robust losses allows us to quickly regain robustness to outliers in the case of the M3500 and SubT datasets, but a simple gradient descent method may still fail

<sup>13</sup> For instance, RANSAC (Section 3.2.1) optimizes the maximum consensus loss via sampling (an option that is only viable for the low-dimensional optimization problems arising in the front-end), while graduated non-convexity allows optimizing a broad variety of losses including the truncated quadratic loss (more on this in Section 3.3.4 below).

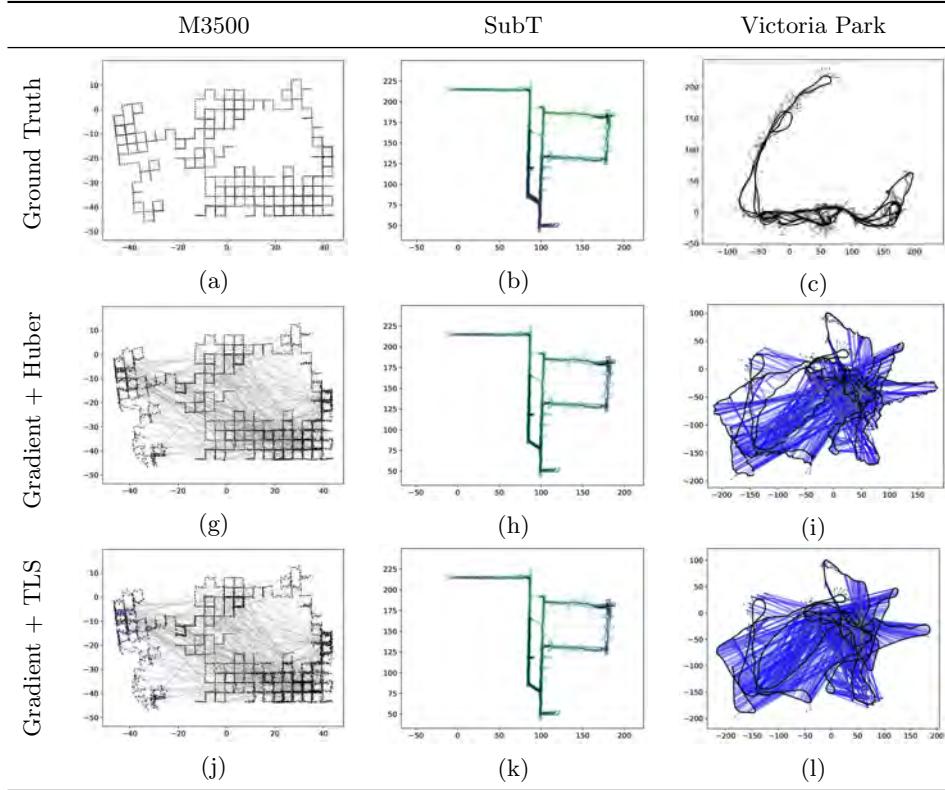


Figure 3.6 Solving SLAM problems with outliers using robust loss functions and gradient descent: (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using gradient descent and Huber loss. (h)-(l) Trajectory estimates obtained using gradient descent and truncated quadratic loss. Measurements are visualized as colored edges. In particular, an edge is colored in gray if it is correctly classified as inlier or outlier by the optimization (*i.e.*, an inlier that falls in the quadratic region of the Huber or truncated quadratic loss); it is colored in red if it is an outlier incorrectly classified as an inlier (a “false positive”); it is colored in blue if it is an inlier incorrectly classified as an outlier (a “false negative”).

to correctly optimize heavily non-convex functions such as the truncated quadratic cost, as in the case of the Victoria Park dataset. We will address this issue with a better solver, based on *graduated non-convexity*, below. Moreover, while gradient descent already improves performance in many of the instances, as shown in Figure 3.6, it has slow convergence tails. For instance, in our experiments, it often takes thousands of iterations to converge. Therefore, in the rest of this section we discuss more advanced solvers, that improve both convergence quality and speed.

As a concluding remark before delving into more advanced solvers, we observe that while it might seem that we gave up on our probabilistic framework when

switching to robust loss functions, it is actually possible to derive several robust losses by applying MAP estimation to heavy-tailed noise distributions. For instance, the truncated quadratic loss results from MAP estimation when assuming the noise to follow a max-mixture distribution between a Gaussian density (describing the inliers) and a uniform distribution (describing the outliers) [34].

### 3.3.1 Iteratively Reweighted Least Squares

M-Estimation replaces the least-squares loss by a robust loss  $\rho$  in (3.9) — a function that grows sub-quadratically for large residuals. This comes with two prices. First, we lose the efficient solutions already developed for least-squares formulations; for instance, the Gauss-Newton and the Levenberg-Marquardt methods are designed for least squares problems. Second, due to the typical non-convex landscape of M-Estimation, iterative solvers (*e.g.*, based on gradient descent) are sensitive to the quality of initialization and often converge to undesired suboptimal estimates. In this section, we introduce a popular algorithm for solving M-Estimation called *iteratively reweighted least squares* (*IRLS*) which, as the name suggests, allows reusing the efficient least-squares solvers. Then, in the next section, we introduce graduated non-convexity as a technique to improve the convergence of IRLS.

The basic idea behind IRLS is to optimize (3.9) by solving a *weighted* least squares problem at each iteration

$$\mathbf{x}^{(t+1)} = \arg \min_{\mathbf{x}} \sum_i w_i(\mathbf{x}^{(t)}) r_i^2(\mathbf{x}), \quad (3.11)$$

where the weights  $w_i$ 's depend on the estimate  $\mathbf{x}^{(t)}$  from the last iteration. We wish the iterative solutions  $\mathbf{x}^{(t)}$  to converge to the optimal solution of M-Estimation (3.9). This implies that the gradient of the robust loss  $\rho$ , shown in (3.10), must match the gradient of the loss in (3.11). By writing down the gradient of (3.11) as

$$\sum_i 2w_i(\mathbf{x}^{(t)}) r_i(\mathbf{x}) \frac{\partial r_i(\mathbf{x})}{\partial \mathbf{x}}$$

and comparing it to (3.10), we obtain the IRLS weight update rule

$$w_i(\mathbf{x}^{(t)}) = \frac{1}{2r_i(\mathbf{x}^{(t)})} \frac{\partial \rho(r_i(\mathbf{x}^{(t)}))}{\partial r_i(\mathbf{x}^{(t)})} = \frac{\psi(r_i(\mathbf{x}^{(t)}))}{2r_i(\mathbf{x}^{(t)})}, \quad (3.12)$$

where we recall that  $\psi(r_i) := \frac{\partial \rho(r_i)}{\partial r_i}$  is the influence function. Therefore, IRLS alternates computing the weights  $w_i(\mathbf{x}^{(t)})$  for each measurement  $i$ , with performing an optimization step (*i.e.*, a Gauss-Newton or Levengberg-Marquardt iteration) on the weighted least squares problem (3.11).

Figure 3.7 shows the performance of IRLS on the M3500, SubT, and Victoria Park datasets. IRLS converges in tens of iterations and is typically much faster than gradient descent; for instance, gradient descent requires around 5 seconds to

optimize the Huber loss in our M3500 experiments, while IRLS takes less than 1.5 seconds. On the other hand, this faster convergence often comes at the cost of a slightly decreased accuracy, as can be seen by comparing Figure 3.7 and Figure 3.6. The convergence properties of the update rule (3.12) has been studied in [10, 814].

### 3.3.2 Black-Rangarajan Duality

The weight update rule (3.12) is widely used in practice, but its derivation was somewhat heuristic. It also has the issues that (3.12) is not well-defined at the non-differentiable points of  $\rho$  (*e.g.*, the cut-off point of the truncated quadratic loss). We now introduce a more principled framework, namely the *Black-Rangarajan (B-R) duality* [85], to solve M-Estimation using IRLS.

Let us present the intuition of B-R duality using the truncated quadratic loss

$$\rho(r_i(\mathbf{x})) := \min\{r_i^2(\mathbf{x}), \beta_i^2\}, \quad (3.13)$$

where  $\beta_i^2$  is a bound on the  $i$ -th residual such that the  $i$ -th measurement is an inlier if  $r_i^2(\mathbf{x}) \leq \beta_i^2$  and an outlier otherwise. We observe that the cost in (3.13) can be equivalently written as a sum of two terms by introducing a new weight variable  $w_i \in [0, 1]$

$$\rho(r_i(\mathbf{x})) := \min_{w_i \in [0,1]} w_i r_i^2(\mathbf{x}) + (1 - w_i) \beta_i^2, \quad (3.14)$$

where the first term is exactly the weighted least squares, and the second term is a function of  $w_i$  that does not depend on  $\mathbf{x}$ . With (3.14), the M-Estimation problem (3.9) with a truncated quadratic loss can be reformulated as

$$\min_{\substack{\mathbf{x} \\ w_i \in [0,1], i=1,\dots,N}} \sum_i [w_i r_i^2(\mathbf{x}) + (1 - w_i) \beta_i^2], \quad (3.15)$$

where we have introduced one  $w_i$  for each measurement residual  $r_i(\mathbf{x})$ . Problem (3.15) is easy to interpret:  $w_i = 1$  implies  $r_i^2(\mathbf{x}) \leq \beta_i^2$  and the  $i$ -th measurement is an inlier;  $w_i = 0$  implies  $r_i^2(\mathbf{x}) > \beta_i^2$  and the  $i$ -th measurement is an outlier. Moreover, all the residuals with  $w_i = 0$  are effectively discarded from the optimization (3.15) and hence robustness is ensured.

B-R duality generalizes the derivation above to a family of robust losses.

**Theorem 3.4** (Black-Rangarajan Duality [85]) *Given a robust loss function  $\rho(\cdot)$ , define  $\phi(z) := \rho(\sqrt{z})$ . If  $\phi(z)$  satisfies  $\lim_{z \rightarrow 0} \phi'(z) = 1$ ,  $\lim_{z \rightarrow \infty} \phi'(z) = 0$ , and  $\phi''(z) < 0$ , then the M-Estimation problem (3.9) is equivalent to*

$$\min_{\substack{\mathbf{x} \\ w_i \in [0,1], i=1,\dots,N}} \sum_{i=1}^N [w_i r_i^2(\mathbf{x}) + \Phi_\rho(w_i)], \quad (3.16)$$

where  $w_i \in [0, 1], i = 1, \dots, N$  are weight variables associated to each residual  $r_i$ ,

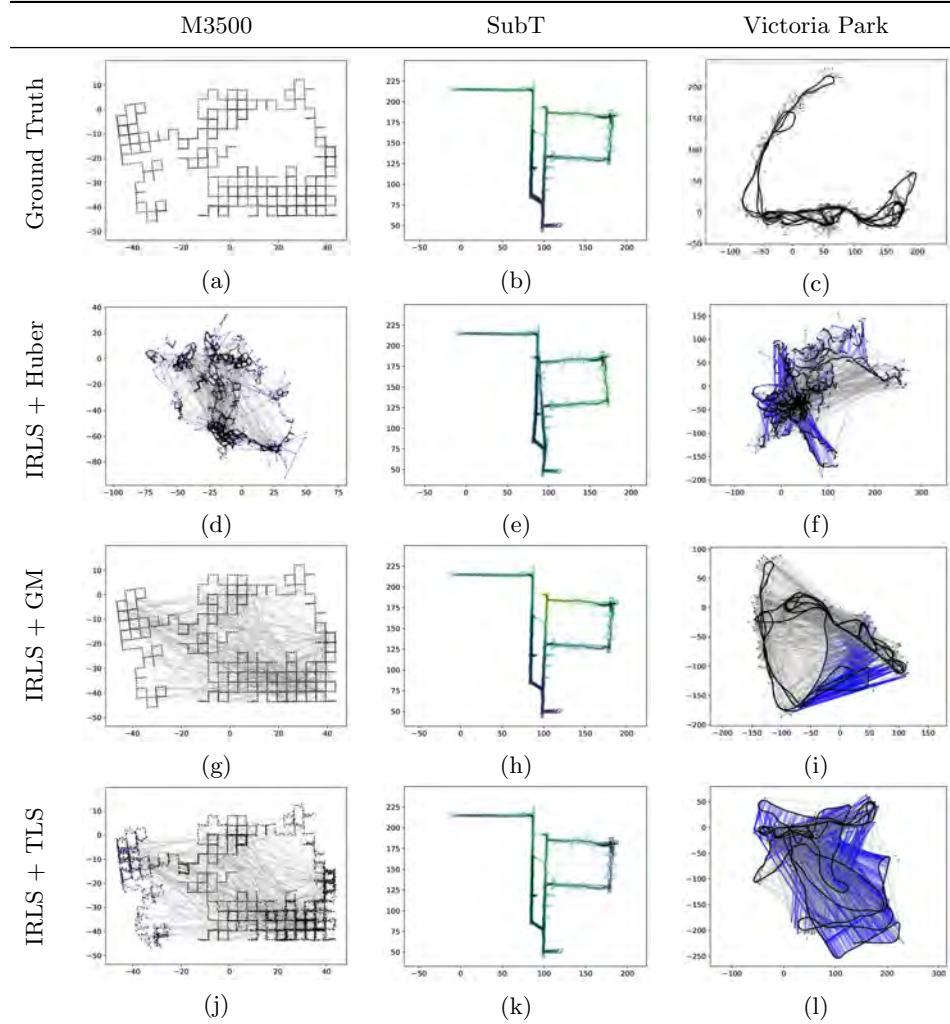


Figure 3.7 Solving SLAM problems with outliers using robust loss functions and Iteratively Reweighted Least-Squares (IRLS): (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using the Huber loss. (g)-(i) Trajectory estimates obtained using the Geman-McClure loss. (j)-(l) Trajectory estimates obtained using the truncated quadratic loss. Measurements are visualized as colored edges. In particular, an edge is colored in gray if it is correctly classified as inlier or outlier by the optimization (*i.e.*, an inlier that falls in the quadratic region of the Huber or truncated quadratic loss); it is colored in red if it is an outlier incorrectly classified as an inlier (a “false positive”); it is colored in blue if it is an inlier incorrectly classified as an outlier (a “false negative”).

and the function  $\Phi_\rho(w_i)$ , referred to as an outlier process, defines a penalty on the weight  $w_i$  whose form is dependent on the choice of robust loss  $\rho$ .

In the case of  $\rho$  being the truncated quadratic loss, we easily derived from (3.14) that  $\Phi_\rho(w_i) = (1-w_i)\beta_i^2$ . When  $\rho$  takes other forms, [85] provides a recipe to derive  $\Phi_\rho(w_i)$ . We give an example for the Geman-McClure (G-M) robust loss.

**Example 3.5** (B-R Duality for G-M Loss) Consider the G-M robust loss function

$$\rho(r_i(\mathbf{x})) = \frac{\beta_i^2 r_i^2(\mathbf{x})}{\beta_i^2 + r_i^2(\mathbf{x})}, \quad (3.17)$$

where  $\beta_i^2$  is a noise bound for the  $i$ -th residual similar to (3.13). The outlier process associated to (3.17) is

$$\Phi_\rho(w_i) = \beta_i^2(\sqrt{w_i} - 1)^2. \quad (3.18)$$

To verify the correctness of (3.18), consider

$$\min_{w_i \in [0,1]} w_i r_i^2(\mathbf{x}) + \Phi_\rho(w_i), \quad (3.19)$$

whose optimal solution is (via setting the gradient of (3.19) to zero)

$$w_i^* = \left( \frac{\beta_i^2}{r_i^2(\mathbf{x}) + \beta_i^2} \right)^2. \quad (3.20)$$

Plugging (3.20) back to the objective of (3.19) recovers the G-M robust loss (3.17).

### 3.3.3 Alternating Minimization

With the introduction of B-R duality, the IRLS algorithm naturally comes out using a common optimization strategy called *alternating minimization* [1109, 80]. The idea is that, although it is difficult to jointly optimize both  $\mathbf{x}$  and  $w_i \in [0, 1]$ ,  $i = 1, \dots, N$  in (3.16), optimization of either  $\mathbf{x}$  or  $w_i$ 's when fixing the other is easy. To see this, observe that when  $w_i$ 's are fixed, problem (3.16) becomes a weighted least squares; analogously, when  $\mathbf{x}$  is fixed, problem (3.16) becomes

$$\min_{w_i \in [0,1], i=1, \dots, N} \sum_{i=1}^N \Phi_\rho(w_i) + w_i r_i^2(\mathbf{x}),$$

which splits into  $N$  subproblems, each optimizing a scalar  $w_i$

$$\min_{w_i \in [0,1]} \Phi_\rho(w_i) + w_i r_i^2(\mathbf{x}). \quad (3.21)$$

Problem (3.21) is easy to solve and often admits a closed-form solution. In fact, for the G-M robust loss, the solution of (3.21) is just (3.20). For the truncated quadratic loss, problem (3.21) reads

$$\min_{w_i \in [0,1]} (1-w_i)\beta_i^2 + w_i r_i^2(\mathbf{x})$$

and admits a closed-form solution

$$w_i^* = \begin{cases} 1 & \text{if } r_i^2(\mathbf{x}) < \beta_i^2 \\ 0 & \text{if } r_i^2(\mathbf{x}) > \beta_i^2 \\ [0, 1] & \text{otherwise} \end{cases}$$

In summary, the  $t$ -th iteration of IRLS in the context of B-R duality alternates between two steps

- 1 **Variable update:** solve a weighted least squares problem using the current weights  $w_i^{(t)}$

$$\mathbf{x}^{(t)} \in \arg \min_{\mathbf{x}} \sum_i w_i^{(t)} r_i^2(\mathbf{x}). \quad (3.22)$$

- 2 **Weight update:** update the weights using  $\mathbf{x}^{(t)}$

$$w_i^{(t+1)} \in \arg \min_{w_i \in [0, 1]} \Phi_\rho(w_i) + w_i r_i^2(\mathbf{x}^{(t)}), i = 1, \dots, N. \quad (3.23)$$

The weight update rule obtained via B-R duality actually matches the popular weight update rule (3.12). Interestingly, instantiating the above IRLS algorithm in SLAM using the G-M robust loss leads to the *dynamic covariance scaling* algorithm [11], which has been proposed in the context of outlier-robust SLAM.

### 3.3.4 Graduated Non-Convexity

The previous section leveraged Black-Rangarajan duality and alternating minimization to derive the IRLS framework that alternates in solving (3.22) and (3.23). However, due to the non-convexity of common robust losses, the convergence of the IRLS framework can be highly sensitive to the quality of initialization, *i.e.*, how close is  $\mathbf{x}^{(0)}$  to the optimal solution of (3.9) or how well does  $w_i^{(0)}$  reflect the inlier-outlier membership of each measurement. For example, [1003] showed that IRLS with the truncated quadratic loss and the Geman-McClure loss might fail when there are as little as 10% outliers in the measurements (*cf.* also with our results in Figure 3.7).

#### *Graduated Non-Convexity*

In this section, we introduce the *Graduated Non-Convexity (GNC)* algorithms, that can make IRLS significantly less sensitive to the quality of initialization. Given a robust cost function  $\rho$ , the basic idea of GNC is to create a smooth version of  $\rho$ , denoted as  $\rho_\mu$ , using a scalar smoothing factor  $\mu$ . Tuning  $\mu$  controls the amount of non-convexity in  $\rho_\mu$ :  $\rho_\mu$  is convex at one end of the spectrum and recovers the original  $\rho$  at the other end of the spectrum. Let us illustrate this using two examples.

**Example 3.6** (GNC Truncated Quadratic Loss) Consider the GNC truncated

quadratic loss function

$$\rho_\mu(r_i(\mathbf{x})) = \begin{cases} r_i^2(\mathbf{x}) & \text{if } r_i^2(\mathbf{x}) \in \left[0, \frac{\mu}{\mu+1}\beta_i^2\right] \\ 2\beta_i|r_i(\mathbf{x})|\sqrt{\mu(\mu+1)} - \mu(\beta_i^2 + r_i^2(\mathbf{x})) & \text{if } r_i^2(\mathbf{x}) \in \left[\frac{\mu}{\mu+1}\beta_i^2, \frac{\mu+1}{\mu}\beta_i^2\right] \\ \beta_i^2 & \text{if } r_i^2(\mathbf{x}) \in \left[\frac{\mu+1}{\mu}\beta_i^2, +\infty\right]. \end{cases} \quad (3.24)$$

$\rho_\mu$  is convex for  $\mu$  approaching zero and retrieves the truncated quadratic loss in (3.13) for  $\mu$  approaching infinity.

**Example 3.7** (GNC Geman-McClure Loss) Consider the GNC Geman-McClure loss function

$$\rho_\mu(r_i(\mathbf{x})) = \frac{\mu\beta_i^2 r_i^2(\mathbf{x})}{\mu\beta_i^2 + r_i^2(\mathbf{x})}. \quad (3.25)$$

$\rho_\mu$  is convex for  $\mu$  approaching  $\infty$  and recovers the G-M loss (3.17) when  $\mu = 1$ .

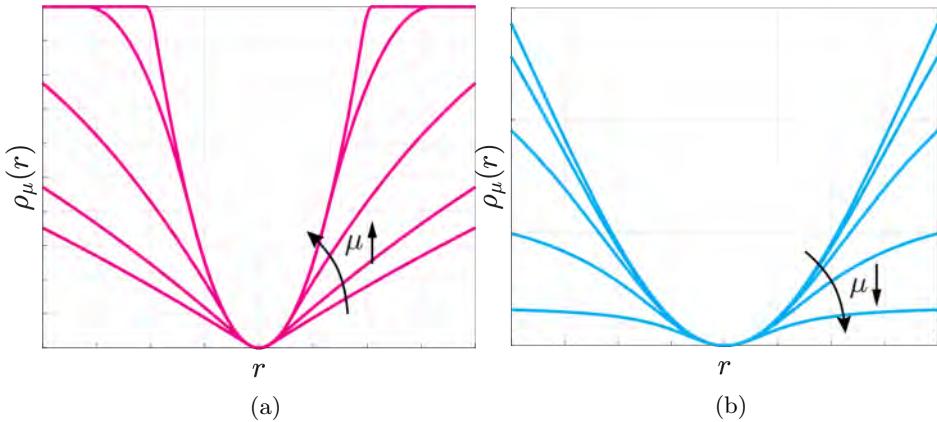


Figure 3.8 Graduated Non-Convexity with control parameter  $\mu$  for (a) Truncated Quadratic loss and (b) Geman-McClure loss. [1219] (©2020 IEEE)

Figure 3.8(a) and (b) plot the GNC truncated quadratic loss and the GNC Geman-McClure loss, respectively. Observe how increasing or decreasing the control parameter  $\mu$  adds more non-convexity to the function.

One nice property of the smoothed GNC functions in (3.24) and (3.25) is that the B-R duality still applies. For the GNC truncated quadratic loss (3.24), applying B-R duality leads to the outlier process

$$\Phi_{\rho_\mu}(w_i) = \frac{\mu(1-w_i)}{\mu+w_i}\beta_i^2.$$

For the GNC Geman-McClure loss (3.25), applying B-R duality leads to the outlier

process

$$\Phi_{\rho_\mu}(w_i) = \mu \beta_i^2 (\sqrt{w_i} - 1)^2.$$

We are now ready to state the GNC algorithm, which at each iteration performs three steps

- 1 **Variable update:** solve a weighted least squares problem using the current weights  $w_i^{(t)}$

$$\mathbf{x}^{(t)} \in \arg \min_{\mathbf{x}} \sum_i w_i^{(t)} r_i^2(\mathbf{x}). \quad (3.26)$$

- 2 **Weight update:** update the weights using  $\mathbf{x}^{(t)}$

$$w_i^{(t+1)} \in \arg \min_{w_i \in [0,1]} \Phi_{\rho_\mu}(w_i) + w_i r_i^2(\mathbf{x}^{(t)}), i = 1, \dots, N. \quad (3.27)$$

- 3 **Control parameter update:** Increase or decrease  $\mu$  to add more nonconvexity to  $\rho_\mu$ .

The GNC algorithm is similar to the IRLS algorithm, except that it starts with a convex, smoothed function  $\rho_\mu$  and then iteratively updates the control parameter  $\mu$  to gradually add more non-convexity to  $\rho_\mu$  to approach the original loss function  $\rho$ . Depending on the definition of the smoothed loss  $\rho_\mu$ , one would recover the original  $\rho$  by either increasing or decreasing  $\mu$ . For instance, the smoother GNC truncated quadratic loss recovers the original truncated quadratic loss when  $\mu$  is large, hence  $\mu$  is increased by a constant factor  $\gamma > 1$  at each GNC iteration (*e.g.*,  $\gamma = 1.4$  in [1219]). Conversely, the smoother Geman-McClure loss recovers the original GM loss when  $\mu$  is close to 1, hence  $\mu$  is *divided* by  $\gamma > 1$  at each GNC iteration.

Figure 3.9 showcases the SLAM trajectories obtained by applying GNC on the same three datasets of Figure 3.6, with two different robust losses: the Geman-McClure loss and the truncated quadratic loss. We implemented GNC using GT-SAM’s GNCOptimizer. By comparing the figure with Figure 3.6 and Figure 3.7 we observe that GNC ensures better convergence (*i.e.*, it is less prone to being stuck in local minima) and recovers fairly accurate trajectories in all the three datasets. While GNC has been shown to be extremely resilient to outliers (*e.g.*, it has shown to tolerate around 80-90% incorrect loop closures in real-world problems [1219, 165]), we remark that the approach does not provide any convergence guarantees. Moreover, its performance has been empirically seen to be problem-dependent, and while it leads to superb performance in pose-graph optimization problems, its performance largely degrades in other perception problems [1002].

Below we showcase a problem when GNC fails and also show that combining front-end and back-end outlier rejection can be beneficial. Towards this goal, we are going to consider a slightly more challenging SLAM setup compared to the ones discussed above. Earlier in this chapter, we considered pose-graph optimization problems (*e.g.*, Figure 3.1) where the odometry is reliable but there might be

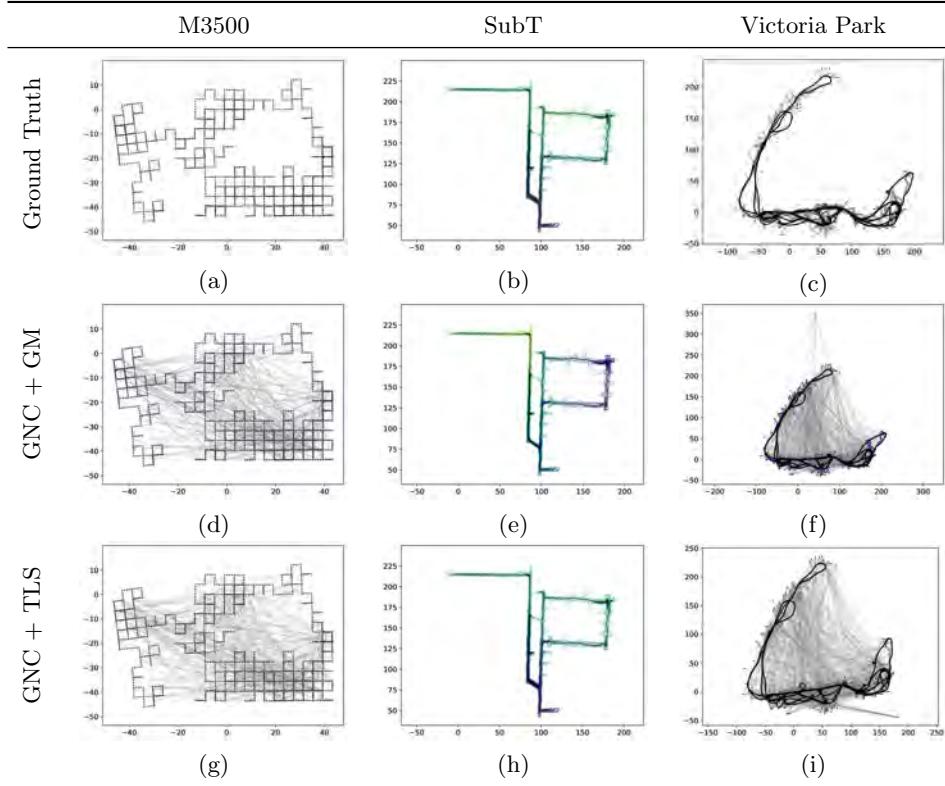


Figure 3.9 Solving SLAM problems with outliers using robust loss functions and graduated non-convexity (GNC): (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using the Geman-McClure loss. (g)-(i) Trajectory estimates obtained using the truncated quadratic loss. Measurements are visualized as colored edges. In particular, an edge is colored in gray if it is correctly classified as inlier or outlier by the optimization (*i.e.*, an inlier that falls in the quadratic region of the Huber or truncated quadratic loss); it is colored in red if it is an outlier incorrectly classified as an inlier (a “false positive”); it is colored in blue if it is an inlier incorrectly classified as an outlier (a “false negative”).

outliers in the loop closures or in the landmark measurements. This setting essentially assumes the presence of an “odometry backbone” that largely simplifies the problem by providing a set of trusted measurements while also allowing building an initial guess for the robot poses. While this assumption is realistic in many problems,<sup>14</sup> certain SLAM applications might lack an odometry backbone. For in-

<sup>14</sup> The fact that odometric measurements are computed between consecutive frames makes data association relatively simpler, in particular when the rate of the sensor (*e.g.*, camera framerate) is much faster than the motion of the robot. Intuitively, consecutive frames will provide snapshots of the scene from similar viewpoints, thus reducing sources of errors for feature matching, including illumination changes, occlusions, lack of viewpoint invariance, or perceptual aliasing.

stance, certain odometry sensors might produce incorrect measurements, *e.g.*, due to wheel slippage in wheel odometry, or incorrect lidar alignment in lidar odometry. Another example arises in multi-robot SLAM, where each robot has an odometry backbone, but the overall SLAM problem (including the trajectory of all robots) does not [725].

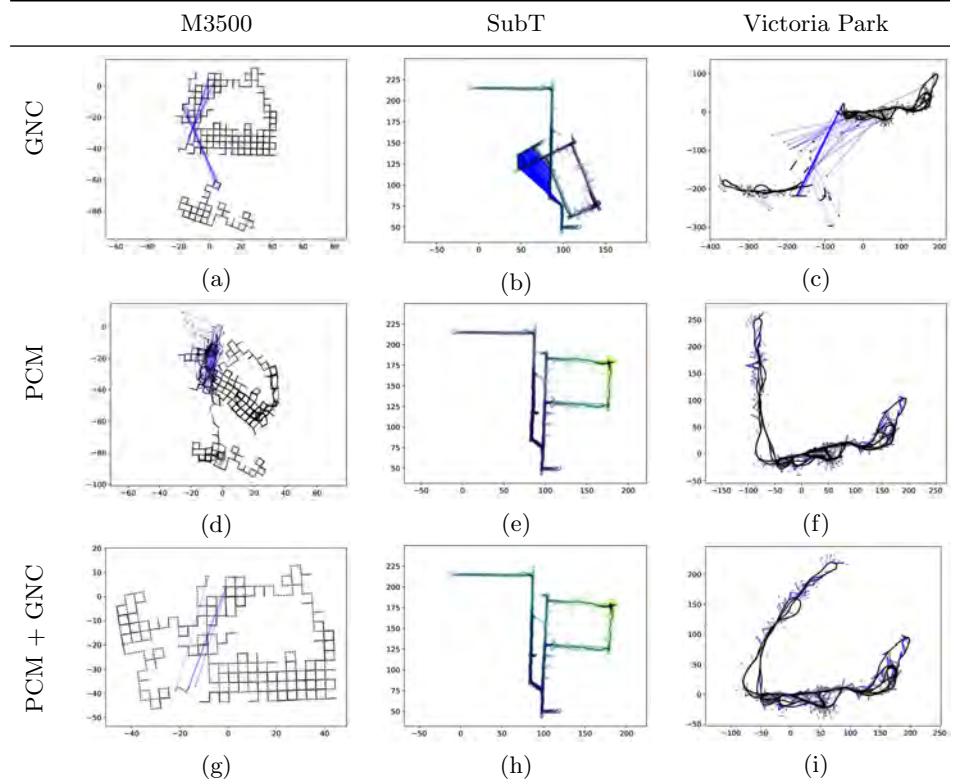


Figure 3.10 SLAM problems with outliers in both the loop closures and landmark measurements, as well as the odometry: (a)-(c) Trajectory estimates obtained using GNC with the truncated quadratic loss. (d)-(f) Trajectory estimates obtained using PCM for front-end outlier rejection followed by least squares optimization. (g)-(i) Trajectory estimates obtained using PCM for front-end outlier rejection followed by GNC with truncated quadratic loss.

Robustly solving SLAM problems where both odometry and loop closure measurements can be outliers is extremely hard. To showcase the shortcomings of the approaches we discussed, in this more complex setting with potentially incorrect odometry measurements, we first modify the pose-graph problems in Figure 3.1 by corrupting two randomly selected odometry measurements. Then we attempt to solve the problem with GNC. In particular, in GNC we fix all odometry measure-

ments except the two potentially corrupted measurements as inliers. Figure 3.10 shows the trajectories obtained by solving the problem with GNC. GNC fails due to the corrupted initialization and converges to a local minimum by categorizing all measurements (including inliers) as outliers (blue edges). The same figure also shows the result of using PCM to filter out gross outliers before using a least-squares back-end. PCM is agnostic to the initial guess, hence is able to converge to better solutions in the case of the SubT and Victoria Park datasets (Figure 3.10 (e) and (f)). Interestingly, we get best results in the SubT and Victoria Park datasets by combining front-end outlier rejection (PCM) with GNC. At the same time, all three methods (GNC, PCM, and PCM+GNC) fail to converge to acceptable solutions in the M3500 dataset, confirming the hardness of this SLAM setup and the limitations of existing SLAM algorithms in terms of outlier rejection.

### 3.4 Further Readings & Recent Trends

**Consensus Maximization.** While in this chapter we discussed the most basic instantiation of a RANSAC algorithm (according to the initial proposal in [328]), it is worth mentioning that the literature offers many RANSAC variants, including variants that refine estimates through local optimization [216], use better scores (rather than the size of the consensus set) in the RANSAC iterations (*e.g.*, MLESAC [869]), or bias the sampling in the RANSAC iterations (*e.g.*, PROSAC [215]). The recent literature also includes differentiable variants of RANSAC [1171] and variants that attempt to find the inliers when the parameter  $\gamma$  in (3.4) is unknown (*e.g.*, [46]). A recent survey and evaluation of RANSAC variants can be found in [1106].

Beyond RANSAC, the literature also includes approaches for *exact* consensus maximization, typically based on *branch-and-bound* [64, 436, 1287, 649, 1023, 203, 507, 132, 1223, 1221]. Despite its global optimality guarantees, branch-and-bound has exponential runtime in the worst case and does not scale to high-dimensional problems.

**Pairwise Consistency Maximization.** The PCM approach described in Section 3.2.2 was originally proposed by Mangelson et al. [725] in the context of multi-robot SLAM, where this approach showed particular promise. Graph-theoretic outlier rejection has also been investigated in computer vision. San Segundo and Artieda [964] build an association graph and find the maximum clique for 2D image feature matching. Perera and Barnes [866] segment objects under rigid-body motion with a clique formulation. Leordeanu and Hebert [637] establish image matches by finding strongly-connected clusters in the correspondence graph with an approximate spectral method. Enqvist et al. [309] develop an outlier rejection algorithm for 3D-3D and 2D-3D registration based on approximate vertex cover. Yang and Carlone [1214], Yang et al. [1220] and Bustos et al. [133] investigate graph-theoretic outlier rejection based on maximum clique for 3D-3D registration. The idea of checking consistency across a subset of measurements also arises in Latif

et al. [626], which perform loop closure outlier rejection by clustering measurements together and checking for consistency using a Chi-squared-based test. The PCM paper [725], similarly to the discussion in this chapter, focuses on pairwise consistency. More recently, PCM has been extended to group-k consistency (*i.e.*, the case where the consistency constraint (3.7) involves  $k$  measurements instead of only 2 measurements) in [1002, 1003, 332]. These papers essentially generalize the notion of consistency graphs to consistency *hypergraphs*, where each hyper-edge involves  $k$  nodes. Related work also considers soft variations of the maximum clique problem, where the binary condition (3.7) is relaxed to produce continuous weights on the edges of the consistency graph [709, 708]. These methods have been used in practical applications, including subterranean exploration [298], lidar point-cloud localization [710], multi-robot metrics-semantic mapping [1092], and global localization in unstructured environments [33].

**Alternating Minimization and Graduated Non-Convexity.** M-estimation has been a popular approach for robust estimation in robotics [99] and vision [975, 173]. Tavish and Barfoot [1072] investigate the performance of different loss functions. Several papers investigate formulations with auxiliary variables as the one in (3.16), without realizing the connection to M-estimation provided by the Black-Rangarajan duality (Theorem 3.4). For instance, Sünderhauf and Protzel [1056], Sünderhauf and Protzel [1055] and Agarwal et al. [11] augment the problem with latent binary variables responsible for deactivating outliers. Lee et al. [635] use expectation maximization. Olson and Agarwal [824] use a max-mixture distribution to approximate multi-modal measurement noise. Recently, Barron [57] proposes a single parametrized function that generalizes a family of robust loss functions in M-estimation. Chebrolu et al. [174] design an expectation-maximization algorithm to simultaneously estimate the unknown quantity  $\boldsymbol{x}$  and choose a suitable robust loss function  $\rho$ . The graduated non-convexity algorithm was first introduced in [86, 85] for outlier rejection in early vision applications; more recently, the algorithm was used for point cloud registration [1294, 1220], SLAM [1219], and other applications [34]. Recently, Peng et al. [857] have proposed an algorithm similar to GNC and IRLS, that is based on the idea of *smooth majorization* in optimization and can be applied to a broad set of robust losses. Moreover, Peng et al. [857] derives global and local convergence guarantees for GNC.

**Certifiable Algorithms.** The algorithms described so far can be broadly divided into two categories: (i) *fast heuristics* (*e.g.*, RANSAC or local solvers for M-estimation), which are efficient but provide little performance guarantees, and (ii) *global solvers* (*e.g.*, branch-and-bound), which offer optimality guarantees but scale poorly with the problem size. Recent years have seen the advent of a new type of methods, called *certifiable algorithms*, that try to strike a balance between tractability and optimality. Certifiable algorithms relax non-convex robust estima-

tion problems into convex *semidefinite programs* (SDP),<sup>15</sup> whose solutions can be obtained in polynomial time and provide readily checkable *a posteriori* global optimality certificates. Certifiable algorithms for robust estimation have been proposed in the context of rotation estimation [1215], 3D-3D registration [1220], and pose-graph optimization [616, 153]. A fairly general approach to derive certifiable algorithms for problems with outliers is described in [1216, 1218], while connections with parallel work in statistics is discussed in [147]. With few notable exceptions, these algorithms, albeit running in polynomial time, are still computationally expensive and typically much slower than heuristics methods. In some cases, the insights behind these algorithms can be used to certify optimality of a solution obtained with a fast heuristic [1216], hence getting the best of both worlds.

<sup>15</sup> We are going to review the notion of certifiable algorithms in the context of SLAM in Chapter 6.

## 4

### Differentiable Optimization

Chen Wang, Krishna Murthy Jatavallabhula, and Mustafa Mukadam

As presented in Chapter I, the design of a contemporary SLAM system generally adheres to a front-end and back-end architecture. In this structure, the front-end is typically responsible for pre-processing sensor data and generating an initial estimate of the robot’s trajectory and the map of the environment, while the back-end refines these initial estimates to improve overall accuracy. Recent advances in machine learning have provided new approaches, based on deep neural networks, that have the potential to enhance some of the functionalities in the SLAM front-end. For instance, deep learning-based methods can exhibit impressive performance in feature detection and matching [968, 267, 1264] and front-end motion estimation [1164, 1079]. These methods train a neural network from a large dataset of examples, and then make estimations without being explicitly programmed to perform the task. Meanwhile, geometry-based techniques persist as an essential element for the SLAM back-end, primarily due to their generality and effectiveness in producing a globally consistent estimate by solving an optimization problem [1205].

While in principle one could just “plug” a learning-based SLAM front-end in the SLAM architecture and feed the corresponding outputs to the back-end, the use of learning-based techniques opens the door for a less unidirectional information exchange. In particular, the back-end can now provide feedback to the front-end, enabling it to learn directly from the back-end estimates in a way that the two modules can more harmoniously cooperate to reduce the estimation errors. Reconciling geometric approaches with deep learning to leverage their complementary strengths is a common thread in a large body of recent work in SLAM. In particular, an emerging trend is to differentiate through geometry-based optimization problems arising in the SLAM back-end. Intuitively, differentiating through an optimization problem allows understanding how the optimal solution of that problem (*e.g.*, our SLAM estimate) depends on the parameters of that problem —in our case, the measurements produced by a learning-based front-end; this in turns allows optimizing the front-end to maximize the SLAM accuracy. One could think about this as a *bilevel optimization* problem, *i.e.*, an upper-level optimization process subject to a lower-level optimization —in particular, a neural network based-optimization

to train the front-end, subject to a geometry-based optimization that computes the SLAM solution for a given front-end output.

The ability to compute gradients end-to-end through an optimization is the core of solving a bilevel optimization problem, which allows neural models to take advantage of geometric priors captured by the optimization. The flexibility of such a scheme has led to promising state-of-the-art results in a wide range of applications such as structure from motion [1077], motion planning [81, 1213], SLAM [511, 1079], BA [1065, 1264], state estimation [1241, 182], and image alignment [713].

In this chapter, we illustrate the basics of how to differentiate through nonlinear least squares problems, such as the ones arising in SLAM. Specifically, Section 4.1 restates the non-linear least square (NLS) problem. Section 4.2 describes how to differentiate through the NLS problem. Section 4.3 shows how to differentiate problems defined on manifold. Section 4.4 discusses numerical challenges of the above differentiation and introduces related machine learning libraries. Finally, Section 4.5 provides examples of differentiable optimization in contemporary SLAM systems.

## 4.1 Recap on Nonlinear Least Squares

Non-linear least squares (NLS) estimate the parameters of a model by minimizing the sum of the squares of the mismatch between observed values and those predicted by the model. Unlike linear least squares, NLS involve a model that is non-linear in the parameters. Beyond our factors graphs in Chapter 1, this approach is widely used in many fields such as statistics, physics, and engineering, where it is useful for fitting complex models to data when the relationship between variables is not straightforward, enabling more accurate and robust predictions.

Specifically, NLS aim to find variables  $\mathbf{x} \in \mathbb{R}^n$  by solving:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}) = \arg \min_{\mathbf{x}} \frac{1}{2} \sum_i \left\| \underbrace{w_i \mathbf{c}_i(\mathbf{x}_i)}_{\mathbf{r}_i(\mathbf{x}_i)} \right\|^2, \quad (4.1)$$

where the objective  $\mathcal{L}(\mathbf{x})$  is a sum of squared vector-valued residual terms  $\mathbf{r}_i$ , each a function of  $\mathbf{x}_i \subset \mathbf{x}$  that are (non-disjoint) subsets of the optimization variables  $\mathbf{x} = \{\mathbf{x}_i\}$ . While for now we assume  $\mathbf{x}_i$  to be vectors, later in the chapter we generalize the discussion to the case where the variables belong to a manifold. For flexibility, here we represent a residual  $\mathbf{r}_i(\mathbf{x}_i) = w_i \mathbf{c}_i(\mathbf{x}_i)$  as a product of a weight  $w_i$  and vector cost  $\mathbf{c}_i$ .

As explained in Chapter 1, a NLS is normally solved by iteratively linearizing the nonlinear objective around the current variables to get the linear system  $(\sum_i \mathbf{J}_i^\top \mathbf{J}_i) \delta \mathbf{x} = (\sum_i \mathbf{J}_i^\top \mathbf{r}_i)$ , then solving the linear system to find the update  $\delta \mathbf{x}$ , and finally updating the variables  $\mathbf{x} \leftarrow \mathbf{x} + \delta \mathbf{x}$ , until convergence. We have also commented in Chapter 2 that the addition in the update step is more generally a retraction mapping for variables that belong to a manifold. In the linear system,

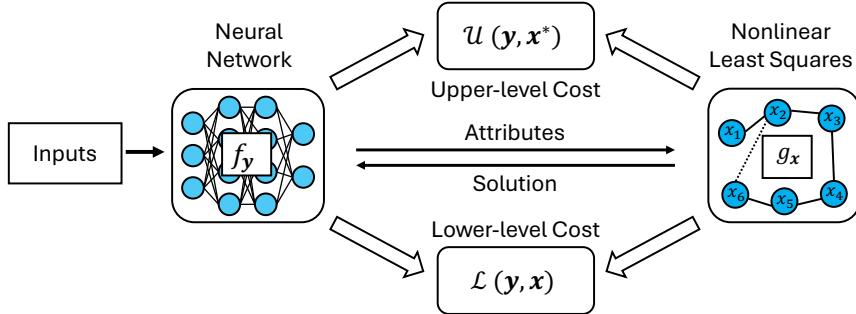


Figure 4.1 A modern SLAM system often involves both neural networks and nonlinear least squares. To eliminate compound errors introduced by optimizing the two modules separately, we can optimize the system in an end-to-end manner by formulating the entire system as a bilevel optimization, which involves an upper-level cost and a lower-level cost.

$\mathbf{J}_i = [\partial \mathbf{r}_i / \partial \mathbf{x}]$  are the Jacobians of residuals with respect to the variables. This iterative method above, called Gauss-Newton (GN), is a nonlinear optimizer that is (approximately) second-order, since  $\sum_i \mathbf{J}_i^\top \mathbf{J}_i$  is an approximation of the Hessian. To improve robustness and convergence, variations like Levenberg-Marquardt (LM) dampen the linear system, while others adjust the step size for the update with line search, e.g., Dogleg introduced in Chapter 1.

## 4.2 Differentiation Through Nonlinear Least Squares

bilevel optimization

To seamlessly merge deep learning with nonlinear least squares, differentiable nonlinear least squares (DNLS) are often required to solve the optimization problem illustrated in Figure 4.1. This necessitates gradients of the solution  $\mathbf{x}^*$  with respect to any upper-level neural model parameters  $\mathbf{y}$  that parameterize the objective  $\mathcal{U}(\mathbf{x}; \mathbf{y})$  and, in turn, any costs  $c_i(\mathbf{x}_i; \mathbf{y})$  or initialization for variables  $\mathbf{x}_{\text{init}}(\mathbf{y})$ . The goal is to learn these parameters  $\mathbf{y}$  end-to-end with a lower-level learning objective  $\mathcal{L}$  defined as a function of  $\mathbf{x}$ . This results in a *Bilevel Optimization* (BLO), which can be written as:

$$\mathbf{y}^* = \arg \min_{\mathbf{y} \in \Theta} \mathcal{U}(\mathbf{y}, \mathbf{x}^*), \quad (4.2a)$$

$$\text{s. t. } \mathbf{x}^* = \arg \min_{\mathbf{x} \in \Psi} \mathcal{L}(\mathbf{y}, \mathbf{x}), \quad (4.2b)$$

where  $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$  is a lower-level (LL) cost,  $\mathcal{U} : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$  is a upper-level (UL) cost,  $\mathbf{x} \in \Psi$  and  $\mathbf{y} \in \Theta$  are the feasible sets.

In practice, the variables  $\mathbf{x}$  are often parameters with explicit physical meanings such as camera poses, while  $\mathbf{y}$  are parameters without physical meanings such as weights in a neural network. We next present two examples to explain this.

**Example 4.1** (Visual SLAM with Learned Features) Imagine a SLAM system that leverages a neural network (parameterized by  $\mathbf{y}$ ) for feature extraction/matching, while utilizing BA for pose estimation (parameterized by  $\mathbf{x}$ ), which take the feature matching as an input. In this example, the UL cost (4.2a) can be feature matching error for optimizing the network, while the LL cost  $L$  (4.2b) can be the reprojection error for BA. Intuitively, the optimal solution  $\mathbf{x}^*$  for the camera poses and landmark positions plays the role of a supervisory signal in the neural network training. Therefore, optimizing the BLO (4.2) allows us to further reduce the matching error via back-propagating the BA reprojection errors [1264].

**Example 4.2** (PGO) Imagine a SLAM system that uses a neural network for front-end pose estimation, while leverages PGO as the back-end to eliminate odometry drifts. In this example, both UL and LL costs can be the pose-graph error. The difference is the UL cost optimizes the network parameterized by  $\mathbf{y}$ , while the LL cost optimizes the camera poses parameterized by  $\mathbf{x}$ . As a result, the front-end network can leverage global geometric knowledge obtained through pose-graph optimization by back-propagating the pose residuals from the back-end PGO [346].

BLO is a long-standing and well-researched problem [1145, 516, 676]. Solving a BLO often relies on gradient-descent techniques. Specifically, the UL optimization performs updates in the form  $\mathbf{y} \leftarrow \mathbf{y} + \delta\mathbf{y}$ , where  $\delta\mathbf{y}$  is a step in the direction of the negative gradient. Therefore, we need to compute the gradient of  $\mathcal{U}$  with respect to the UL variable  $\mathbf{y}$ , which can be written as

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} + \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}, \quad (4.3)$$

where the term  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  involves indirect gradient computation. Since other direct gradient terms in (4.3) are easy to obtain, the challenge of solving a BLO (4.2) is to compute the term  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$ . For this purpose, a series of techniques have been developed from either explicit or implicit perspectives. This involves recurrent differentiation through dynamical systems or implicit differentiation theory, which are often referred to as **unrolled differentiation** and **implicit differentiation**, respectively. These algorithms have been summarized in [676, 1145] and here we list a generic framework incorporating both methods in Algorithm 1. We next introduce unrolled differentiation and implicit differentiation.

### 4.2.1 Unrolled Differentiation

Unrolled Differentiation needs Automatic Differentiation (AutoDiff) through the LL optimization to solve a BLO problem. Specifically, given an initialization  $\mathbf{x}_0 = \Phi_0(\mathbf{y})$  at step  $t = 0$ , the iterative process of unrolled LL optimization is

$$\mathbf{x}_t = \Phi_t(\mathbf{x}_{t-1}; \mathbf{y}), \quad t = 1, \dots, T, \quad (4.4)$$

unrolled differentiation

**Algorithm 1** Solving BLO by *Unrolled Differentiation* or *Implicit Differentiation*.

---

```

1: Initialization:  $\mathbf{y}_0, \mathbf{x}_0$ .
2: while Not Convergent ( $\|\mathbf{y}_{k+1} - \mathbf{y}_k\|$  is large enough) do
3:   Obtain  $\mathbf{x}_T$  by solving (4.2b) by a generic optimizer  $\mathcal{O}$  with  $T$  steps.
4:   Efficient estimation of upper-level gradients in (4.3) via
5:   Unrolled Differentiation:  $\hat{\nabla}_{\mathbf{y}_k} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}_k, \mathbf{x}_T)}{\partial \mathbf{y}_k}$  via AutoDiff in (4.7).
6:   Implicit Differentiation (Algorithm 2): Compute

$$\hat{\nabla}_{\mathbf{y}_k} \mathcal{U} = \left. \frac{\partial \mathcal{U}}{\partial \mathbf{y}_k} \right|_{\mathbf{x}_T} + \left. \frac{\partial \mathcal{U}}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial \mathbf{y}_k} \right|_{\mathbf{x}_T},$$

where the implicit derivatives  $\frac{\partial \mathbf{x}^*}{\partial \mathbf{y}_k}$  can be obtained by solving an equation derived via lower-level optimality conditions (surveyed in following sections).
7:
8:   Compute  $\mathbf{y}_{k+1}$  via gradients using  $\hat{\nabla}_{\mathbf{y}_k} \mathcal{U}$ .
9: end while

```

---

where  $\Phi_t$  denotes an updating scheme based on the LL problem at the  $t$ -th step and  $T$  is the number of iterations. One updating scheme is the gradient descent:

$$\Phi_t(\mathbf{x}_{t-1}; \mathbf{y}) = \mathbf{x}_{t-1} - \eta_t \cdot \frac{\partial \mathcal{L}(\mathbf{x}_{t-1}, \mathbf{y})}{\partial \mathbf{x}_{t-1}}, \quad (4.5)$$

where  $\eta_t$  is a learning rate and the term  $\frac{\partial \mathcal{L}(\mathbf{x}_{t-1}, \mathbf{y})}{\partial \mathbf{x}_{t-1}}$  can be computed from AutoDiff.<sup>1</sup> Therefore, we can compute the  $\nabla_{\mathbf{y}} \mathcal{U}(\mathbf{y})$  by substituting  $\mathbf{x}_T$  approximately for  $\mathbf{x}^*$  and the full unrolled system can be defined as

$$\mathbf{x}^* \approx \mathbf{x}_T = \Phi(\mathbf{y}) = (\Phi_T \circ \dots \circ \Phi_1 \circ \Phi_0)(\mathbf{y}), \quad (4.6)$$

where the symbol  $\circ$  denotes the function composition. As a result, we only need to consider the following problem instead of a bilevel optimization in (4.2):

$$\min_{\mathbf{y} \in \Theta} \mathcal{U}(\mathbf{y}, \Phi(\mathbf{y})), \quad (4.7)$$

which needs to compute  $\frac{\partial \Phi(\mathbf{y})}{\partial \mathbf{y}}$  via AutoDiff instead of calculating (4.3). It is worth noting that there exist two approaches for computing the recurrent gradients, one of which corresponds to backward propagation in a reverse-mode way [854], and the other corresponds to the forward-mode way [925]. We omit the details of the two approaches of AutoDiff and refer the readers to the AutoDiff libraries such as PyTorch [847] for deep learning and PyPose [1142] and Theseus [870] for SLAM. A review of these approaches can also be found in Liu et al. [676].

<sup>1</sup> While here we mention gradient descent, the same ideas can be extended to other iterative optimization methods, such as Gauss-Newton.

### 4.2.2 Truncated Unrolled Differentiation

The reverse and forward modes are two precise recurrent gradient calculation methods but are time-consuming with the full iterative propagation. This is due to the complicated long-term dependencies of the UL problem on  $\mathbf{x}_t$ , where  $t = 0, 1, \dots, T$ . This difficulty is further aggravated when both  $\mathbf{y}$  and  $\mathbf{x}$  are high-dimensional vectors. To overcome this challenge, the truncated unrolled differentiation has been investigated as a way to compute high-quality approximate gradients with significantly less computation time and memory. Specifically, by ignoring the long-term dependencies and approximating the gradient of (4.5) with partial history, *i.e.*, storing only the last  $M$  iterations ( $t = T, T-1, \dots, T-M$ ), we can significantly reduce the time and space complexity. It has been proved by Shaban et al. [989] that using fewer backward steps to compute the gradients could perform comparably to optimization with the exact one, while requiring much less memory and computation.

In case of more stringent computational and memory constraints, truncated unrolled differentiation is still often a bottleneck in modern robotic applications. Therefore, researchers have also tried to further simplify the truncated differentiation by only performing a one-step iteration in (4.4) to remove the recursive structure [673], *i.e.*,

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{y}} + \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}}, \quad (4.8)$$

where the term  $\frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}}$  is a Hessian that can be calculated from (4.5) as

$$\frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}} = -\frac{\partial^2 \mathcal{L}(\mathbf{x}_0, \mathbf{y})}{\partial \mathbf{x}_0 \partial \mathbf{y}}. \quad (4.9)$$

Since calculating a Hessian is time-consuming in some applications, we can resort to numerical solutions that apply small perturbations to the variables  $\mathbf{x}$  and calculate an approximation of the second term in (4.8) as a whole:

$$\frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}} \approx \frac{\frac{\partial \mathcal{L}(\mathbf{x}_0^+, \mathbf{y})}{\partial \mathbf{y}} - \frac{\partial \mathcal{L}(\mathbf{x}_0^-, \mathbf{y})}{\partial \mathbf{y}}}{2\epsilon}, \quad (4.10)$$

where  $\epsilon$  is a small scalar and  $\mathbf{x}_0^\pm = \mathbf{x}_0 \pm \epsilon \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{x}_1}$  is a small perturbation. This bypasses an explicit calculation of the Jacobian  $\frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}}$ . Nevertheless, we need to pay attention to the perturbation model if non-Euclidean variables are involved, *e.g.*, variables belonging to Lie Groups. Fortunately, the AutoDiff of Lie Group for Hessian-vector and Jacobian-vector multiplications are supported in modern libraries, such as PyPose [1142], which will be introduced in Section 4.4.

### 4.2.3 Implicit Differentiation

It is intuitive that the term  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  in (4.3) is dependent on the LL cost (4.2b), thus *implicit differentiation* can be used to derive a solution to the gradient.

In calculus, *implicit differentiation* refers to the method that makes use of the chain rule to differentiate an implicit function. To differentiate an implicit function  $y(x)$ , defined by an equation  $R(x, y) = 0$ , it is not generally possible to solve it explicitly for  $y$  and then differentiate. Instead, one can totally differentiate  $R(x, y) = 0$  with respect to  $x$  and then solve the resulting linear equation for  $\frac{dy}{dx}$  to explicitly get the derivative in terms of  $x$  and  $y$ . For instance, consider an implicit function  $x + y + 5 = 0$ , differentiating it with respect to  $x$  on its both sides gives  $\frac{dy}{dx} + \frac{dx}{dx} + \frac{d}{dx}(5) = 0 \Rightarrow \frac{dy}{dx} + 1 + 0 = 0$ . Solving for  $\frac{dy}{dx}$  gives  $\frac{dy}{dx} = -1$ .

Assume the LL cost  $\mathcal{L}$  is at least twice differentiable w.r.t. both  $\mathbf{y}$  and  $\mathbf{x}$ , then we have  $\frac{\partial \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y})} = 0$  due to the optimality condition where  $\mathbf{x}^*$  is a stationary point. Derive the equation  $\frac{\partial \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y})} = 0$  on both sides w.r.t.  $\mathbf{y}$  giving us

$$\frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}} + \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})} \cdot \frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}} = 0. \quad (4.11)$$

This leads to the indirect gradient  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  as

$$\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}} = - \left( \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})} \right)^{-1} \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}}, \quad (4.12)$$

The strength of (4.12) is that we convert the indirect gradient among the variables  $\mathbf{y}$  and  $\mathbf{x}$  to direct gradients of  $\mathcal{L}$  at the cost of an inversion of a Hessian matrix. However, the weakness is that a Hessian is often too large to compute, thus it is common to solve a linear system leveraging the fast Hessian-vector product.

**Example 4.3** (Cost of Storing and Inverting the Hessian) Assume both UL and LU costs involve a network with merely 1 million ( $10^6$ ) parameters (32-bit float numbers), thus each network only needs a space of  $10^6 \times 4\text{Byte} = 4\text{MB}$  to store, while their Hessian matrix needs a space of  $(10^6)^2 \times 4\text{Byte} = 4\text{TB}$  to store. This indicates that a Hessian matrix cannot even be explicitly stored in the memory of a low-power computer, thus directly calculating its inverse is impractical.

Recall that our goal is to compute the gradient in (4.3), substituting (4.12) into (4.3) gives us:

$$\begin{aligned} \nabla_{\mathbf{y}} \mathcal{U} &= \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \underbrace{\frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{x}^*} \left( \underbrace{\frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})}}_{(\mathbf{H}^T)^{-1}} \right)^{-1} \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}}}_{\mathbf{v}^T} . \quad (4.13) \\ &= \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \mathbf{q}^T \cdot \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}} \end{aligned}$$

Then we can solve the linear system  $\mathbf{H}\mathbf{q} = \mathbf{v}$  for  $\mathbf{q}^\top$  by optimizing

$$\mathbf{q}^* = \min \arg_{\mathbf{q}} Q(\mathbf{q}) = \min \arg_{\mathbf{q}} \frac{1}{2} \mathbf{q}^\top \mathbf{H} \mathbf{q} - \mathbf{q}^\top \mathbf{v}, \quad (4.14)$$

using efficient linear solvers such as a simple gradient descent or conjugate gradient method [459]. For gradient descent, we need to compute the gradient of  $Q$  as  $\frac{\partial Q(\mathbf{q})}{\partial \mathbf{q}} = \mathbf{H}\mathbf{q} - \mathbf{v}$ , where  $\mathbf{H}\mathbf{q}$  can be computed using the fast Hessian-vector product, *i.e.*, a Hessian-vector product is the gradient of a gradient-vector product:

$$\mathbf{H}\mathbf{q} = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{x} \partial \mathbf{x}} \cdot \mathbf{q} = \frac{\partial (\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \cdot \mathbf{q})}{\partial \mathbf{x}}, \quad (4.15)$$

where  $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \cdot \mathbf{q}$  is a scalar. This means that the Hessian matrix  $\mathbf{H}$  is not explicitly computed or stored. We summarize the computation of implicit differentiation with linear systems in Algorithm 2. The algorithm using a conjugate gradient is similar.

---

**Algorithm 2** Computing *Implicit Differentiation* via Linear System.

---

- 1: **Input:** The current UL variable  $\mathbf{y}$  and the optimal LL variable  $\mathbf{x}^*$ .
- 2: **Initialization:**  $k = 1$ , learning rate  $\eta$ .
- 3: **while** Not Convergent ( $\|\mathbf{q}_k - \mathbf{q}_{k-1}\|$  is large enough) **do**
- 4:     Perform gradient descent:

$$\mathbf{q}_k = \mathbf{q}_{k-1} - \eta (\mathbf{H}\mathbf{q}_{k-1} - \mathbf{v}), \quad (4.16)$$

where  $\mathbf{H}\mathbf{q}_{k-1}$  is computed via the fast Hessian-vector product.

- 5: **end while**
- 6: Assign  $\mathbf{q} = \mathbf{q}_k$
- 7: Compute  $\nabla_{\mathbf{y}} \mathcal{U}$  in (4.3) as:

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \underbrace{\left( \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{y} \partial \mathbf{x}^*(\mathbf{y})} \cdot \mathbf{q} \right)}_{(\mathbf{H}_{\mathbf{yx}} \cdot \mathbf{q})^\top}^\top, \quad (4.17)$$

where  $\mathbf{H}_{\mathbf{yx}} \cdot \mathbf{q}$  can also be computed efficiently using the Hessian-vector product.

---

**Approximations.** Implicit differentiation is complicated to implement but there is one approximation, which is to ignore the implicit components and only use the direct part  $\hat{\nabla}_{\mathbf{y}} \mathcal{U} \approx \frac{\partial U}{\partial \mathbf{y}} \Big|_{\mathbf{x}^*}$ . This is equivalent to taking the solution  $\mathbf{x}_T$  from the LL optimization as *constants* in the UL problem. Such an approximation is more efficient but introduces an error term

$$\epsilon \sim \left| \frac{\partial U}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial \mathbf{y}} \right|. \quad (4.18)$$

Nevertheless, it is useful when the implicit gradients contain products of *small* second-order derivatives, which depends on the specific NLS problems.

### 4.3 Differentiation on Manifold

Given that the state of a typical SLAM system is bound to evolve on a manifold, optimization on manifolds plays a crucial role in solving back-end SLAM problems. We next derive the Jacobians required to differentiate with respect to variables belonging to Lie groups, which is an essential step for differentiation on the manifold.

#### 4.3.1 Lie Group Derivatives

Since we introduced the basic concepts of Lie group, Lie algebra, and their basic operations (*e.g.*, exponential and logarithmic maps) in Chapter 2, this section will briefly recap those concepts but mainly focus on the definition of their derivatives, which is essential for solving a differentiable optimization problem.

Consider a Lie group's manifold  $\mathcal{M}$ . Each point  $\chi$  on this smooth manifold possesses a unique tangent space, denoted by  $T_\chi \mathcal{M}$ , where the fundamental principles of calculus are valid. The Lie algebra, represented as  $\mathfrak{m}$ , is a vector space that can be locally defined to the point  $\chi$  as  $\mathfrak{m} = T_\chi \mathcal{M}$ . The exponential map  $\exp : \mathfrak{m} \rightarrow \mathcal{M}$  projects elements from the Lie algebra to the Lie group, while the logarithmic map  $\log : \mathcal{M} \rightarrow \mathfrak{m}$  serves as its inverse, establishing a bi-directional relationship:

$$\chi = \exp(\tau^\wedge) \Leftrightarrow \tau^\wedge = \log(\chi), \quad (4.19)$$

where hat  $^\wedge$  is a linear invertible map, and  $\tau^\wedge \in \mathfrak{m}$ . By representing the coordinates within the Lie algebra as vectors  $\tau$  in  $\mathbb{R}^n$ , we can define mappings between vector  $\tau$  and the Lie group  $\chi$ :

$$\chi = \text{Exp}(\tau) \Leftrightarrow \tau = \text{Log}(\chi), \quad (4.20)$$

where we redefined the exponential and logarithm maps to directly use a vector as input and output, respectively.

To calculate derivatives on Lie groups, it is crucial to first understand the relative change between two manifold elements, say  $\chi_1$  and  $\chi_2$ . These changes are quantified by first defining the  $\oplus$  and  $\ominus$  operators, which combine one exponential/logarithmic map and one composition. Because composition is generally non-commutative, these operators have right- and left- versions, depending on the order of the operands. The right operators are:

$$\begin{aligned} \text{right-}\oplus &: \quad \chi_2 = \chi_1 \oplus \tau \triangleq \chi_1 \circ \text{Exp}(\tau), \\ \text{right-}\ominus &: \quad \tau = \chi_2 \ominus \chi_1 \triangleq \text{Log}(\chi_1^{-1} \circ \chi_2). \end{aligned} \quad (4.21)$$

The placement of  $\tau$  on the right-hand side in (4.21) signifies that it is expressed in the local frame at  $\chi_1$ . Conversely, the left operators in (4.22) reflect a global frame perspective:

$$\begin{aligned} \text{left-}\oplus &: \quad \chi_2 = \varepsilon \oplus \chi_1 \triangleq \text{Exp}(\varepsilon) \circ \chi_1, \\ \text{left-}\ominus &: \quad \varepsilon = \chi_2 \ominus \chi_1 \triangleq \text{Log}(\chi_2 \circ \chi_1^{-1}), \end{aligned} \quad (4.22)$$

where  $\boldsymbol{\epsilon}$  is expressed in the global frame. Both  $\boldsymbol{\tau}$  and  $\boldsymbol{\epsilon}$  can be viewed as incremental perturbations to the manifold elements. By using corresponding composition operators  $\oplus$  and  $\ominus$ , the variations are expressed as vectors in the tangent space.

With the right  $\oplus$  and  $\ominus$  operators in place, we use the Jacobian matrix  $\mathbf{J}$  to describe perturbations on manifolds. The Jacobian captures the essence of infinitesimal perturbations  $\boldsymbol{\tau}$  within the tangent space  $\mathfrak{m}$ :

$$\begin{aligned}\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} &\triangleq \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{f(\boldsymbol{\chi} \oplus \boldsymbol{\tau}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\tau}} \\ &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{f(\boldsymbol{\chi} \circ \text{Exp}(\boldsymbol{\tau})) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\tau}} \\ &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{\text{Log}(f(\boldsymbol{\chi})^{-1} \circ f(\boldsymbol{\chi} \circ \text{Exp}(\boldsymbol{\tau})))}{\boldsymbol{\tau}}.\end{aligned}\quad (4.23)$$

Let  $g(\boldsymbol{\tau}) = \text{Log}(f(\boldsymbol{\chi})^{-1} \circ f(\boldsymbol{\chi} \circ \text{Exp}(\boldsymbol{\tau})))$ , then the right Jacobian  $\mathbf{J}_R$  can be expressed as the derivative of  $g(\boldsymbol{\tau})$  at  $\boldsymbol{\tau} = 0$ :

$$\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} = \mathbf{J}_R = \left. \frac{\partial g(\boldsymbol{\tau})}{\partial \boldsymbol{\tau}} \right|_{\boldsymbol{\tau}=0}. \quad (4.24)$$

In this way, the derivatives of  $f(\boldsymbol{\chi})$  with respect to  $\boldsymbol{\chi}$  in the manifold are represented by the Jacobian matrix  $\mathbf{J}_R \in \mathbb{R}^{m \times n}$ , where  $m$  and  $n$  are the dimensions of the Lie groups  $\mathcal{M}$  and  $\mathcal{N}$ , respectively. The right Jacobian matrix performs a linear mapping from the tangent space  $\mathfrak{m}$  to the tangent space  $\mathfrak{n} = T_{f(\boldsymbol{\chi})}\mathcal{N}$ .

Similarly, consider an infinitesimal perturbation  $\boldsymbol{\epsilon} \in T_g\mathcal{M}$  applied to the Lie group element  $\boldsymbol{\chi}$ , the left Jacobian  $\mathbf{J}_L$  can be defined with the left plus and minus operators:

$$\begin{aligned}\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} &\triangleq \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{f(\boldsymbol{\epsilon} \oplus \boldsymbol{\chi}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\epsilon}} \\ &= \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\epsilon}} \\ &= \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{\text{Log}(f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \circ f(\boldsymbol{\chi})^{-1})}{\boldsymbol{\epsilon}} \\ &= \left. \frac{\partial \text{Log}(f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \circ f(\boldsymbol{\chi})^{-1})}{\partial \boldsymbol{\epsilon}} \right|_{\boldsymbol{\epsilon}=0}.\end{aligned}\quad (4.25)$$

The resulting left Jacobian  $\mathbf{J}_L \in \mathbb{R}^{n \times m}$  is also a linear mapping, but in the global tangent space from  $T_g\mathcal{M}$  to  $T_g\mathcal{N}$ .

To delve into the local perturbations around a point  $\boldsymbol{\chi}_1$ , we consider perturbations  $\boldsymbol{\tau}$  as  $\boldsymbol{\tau} = \boldsymbol{\chi} \ominus \boldsymbol{\chi}_1$ , with  $\boldsymbol{\chi}$  being a perturbed version of  $\boldsymbol{\chi}_1$ . The covariance matrices  $\boldsymbol{\Sigma}_{\boldsymbol{\chi}}$  defined on the tangent space are derived using the expectation operator  $\mathbb{E}$ , enabling the representation of uncertainties and their propagation:

$$\boldsymbol{\Sigma}_{\boldsymbol{\chi}} \triangleq \mathbb{E}[\boldsymbol{\tau}\boldsymbol{\tau}^T] = \mathbb{E}[(\boldsymbol{\chi} \ominus \boldsymbol{\chi}_1)(\boldsymbol{\chi} \ominus \boldsymbol{\chi}_1)^T]. \quad (4.26)$$

These covariance matrices facilitate the establishment of Gaussian distributions on the manifold, expressed as  $\chi \sim \mathcal{N}(\chi_1, \Sigma_\chi)$ . It is important to note that the covariance matrices  $\Sigma_\chi$  are defined on the tangent space  $T_{\chi_1}\mathcal{M}$ , which allows the uncertainty in the manifold to be represented by a vector and be propagated in the form of covariance matrices.

**Example 4.4** (Visual-Inertial Rotation Estimation) Consider a robot equipped with an inertial measurement unit (IMU) and a camera. Given noisy observations  $\mathbf{R}_{\text{IMU}}$  and  $\mathbf{R}_{\text{Cam}}$  from both sensors, the orientation of the robot can be estimated by minimizing the discrepancy between the measurements, which can be formulated as a nonlinear least squares problem on the manifold SO(3):

$$\hat{\mathbf{R}} = \arg \min_{\mathbf{R} \in \text{SO}(3)} f(\mathbf{R}, \mathbf{R}_{\text{IMU}}, \mathbf{R}_{\text{Cam}}), \quad (4.27)$$

where  $f(\cdot)$  is the cost function that quantifies the differences between the estimated orientation  $\mathbf{R}$  and the sensor measurements  $\mathbf{R}_{\text{IMU}}$  and  $\mathbf{R}_{\text{Cam}}$ . With the Jacobian matrices in place, the optimization on the manifold SO(3) for the pose estimation can be effectively managed. The cost function  $f(\cdot)$  can be written as:

$$f(\mathbf{R}) = \|\text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R})\|^2 + \|\text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R})\|^2. \quad (4.28)$$

To minimize  $f(\mathbf{R})$ , we need to compute its gradient with respect to  $\mathbf{R}$  on the manifold SO(3). The gradient can be derived using the right Jacobian  $\mathbf{J}_R$  as:

$$\begin{aligned} \nabla f(\mathbf{R}) &= 2 \left( \frac{\partial \text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R})}{\partial \mathbf{R}} \right)^\top \text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R}) \\ &\quad + 2 \left( \frac{\partial \text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R})}{\partial \mathbf{R}} \right)^\top \text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R}). \end{aligned} \quad (4.29)$$

The gradient  $\nabla f(\mathbf{R})$  can be used in conjunction with optimization algorithms like gradient descent which moves along the tangent space and reprojecting back to the manifold to update the pose  $\mathbf{R}$  iteratively:

$$\mathbf{R}_{k+1} = \mathbf{R}_k \text{Exp}(-\alpha \nabla f(\mathbf{R})), \quad (4.30)$$

where  $\alpha$  is the step size. This iterative process continues until the cost function  $f(\mathbf{R})$  converges to a minimum, providing a pose estimate  $\hat{\mathbf{R}}$  that leverages both sensor measurements.

#### 4.3.2 Differentiation Operations on Manifold

For typical manifold operations, we can derive closed-form expressions for the Jacobians associated with inversion, composition, and group actions. These expressions

facilitate a comprehensive approach to optimization in SLAM, by enabling the computation of function derivatives on manifolds with the chain rule:

$$\frac{\partial \mathcal{Z}}{\partial \chi} = \frac{\partial \mathcal{Z}}{\partial \mathcal{Y}} \frac{\partial \mathcal{Y}}{\partial \chi}, \quad (4.31)$$

where  $\mathcal{Z} = g(\mathcal{Y})$ , and  $\mathcal{Y} = f(\chi)$ .

**Jacobians of inversion** can be derived through the application of the function  $f(\chi) = \chi^{-1}$  with (4.23) for the right Jacobian  $\mathbf{J}_R$ , which leads to:

$$\begin{aligned} \frac{\partial \chi^{-1}}{\partial \chi} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi^{-1})^{-1}(\chi \text{Exp}(\tau))^{-1})}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\chi \text{Exp}(\tau)^{-1} \chi^{-1})}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{(\chi(-\tau)^\wedge \chi^{-1})^\vee}{\tau}. \end{aligned} \quad (4.32)$$

**Jacobians of composition** can be derived through the application of the function  $f(\chi) = \chi \circ \chi_1$  with the Equation (4.23). The derivative of the composition operator  $\chi \circ \chi_1$  with respect to  $\chi$  is:

$$\begin{aligned} \frac{\partial(\chi \circ \chi_1)}{\partial \chi} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi \chi_1)^{-1}(\chi \text{Exp}(\tau) \chi_1))}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\chi_1^{-1} \text{Exp}(\tau) \chi_1)}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{(\chi_1^{-1} \tau^\wedge \chi_1)^\vee}{\tau}. \end{aligned} \quad (4.33)$$

The derivative of the composition operator  $\chi \circ \chi_1$  with respect to  $\chi_1$  is:

$$\begin{aligned} \frac{\partial(\chi \circ \chi_1)}{\partial \chi_1} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi \chi_1)^{-1}(\chi \chi_1 \text{Exp}(\tau)))}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\text{Exp}(\tau))}{\tau} \\ &= \mathbf{I}. \end{aligned} \quad (4.34)$$

**Jacobians of the manifold  $\mathcal{M}$**  are characterized by the right Jacobian of  $\chi$  which is derived from the exponential map of  $\tau \in \mathbb{R}^m$ . This is expressed as:

$$\mathbf{J}_r(\tau) \triangleq \frac{\tau \partial \text{Exp}(\tau)}{\partial \tau}. \quad (4.35)$$

The right Jacobian conveys minor changes in  $\tau$  to modifications in the local tangent space at  $\text{Exp}(\tau)$ . Similarly, the left Jacobian of  $\chi$  maps changes of  $\tau$  to variations within the global tangent space of the manifold. This is expressed as:

$$\mathbf{J}_l(\tau) \triangleq \frac{\epsilon \partial \text{Exp}(\tau)}{\partial \tau}. \quad (4.36)$$

**Jacobians of group action** depends on the specific group action set  $v \in \mathcal{V}$ . The group action is defined as:

$$\begin{aligned} J_{\chi}^{\chi \cdot v} &\triangleq \frac{\chi D\chi \cdot v}{D\chi}, \\ J_v^{\chi \cdot v} &\triangleq \frac{v D\chi \cdot v}{Dv}, \end{aligned} \quad (4.37)$$

where  $\chi \in \mathcal{M}$  and  $v \in \mathcal{V}$ .

**Example 4.5** (Robot Arm) Consider a robotic arm with two joints,  $\mathbf{R}_1$  and  $\mathbf{R}_2$ , each represented by an element in  $\text{SO}(3)$ . The final orientation of the robot's end-effector is determined by the composition of the joint rotations:

$$\mathbf{R} = \mathbf{R}_1 \circ \mathbf{R}_2. \quad (4.38)$$

To evaluate the impact of small perturbations  $\boldsymbol{\tau}$  in  $\mathbf{R}_1$  and  $\mathbf{R}_2$  on the end-effector orientation  $\mathbf{R}$ . It can be quantified using the Jacobians of composition:

$$\begin{aligned} \frac{\partial(\mathbf{R}_1 \circ \mathbf{R}_2)}{\partial \mathbf{R}_1} &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{(\mathbf{R}_2^{-1} \boldsymbol{\tau}^\wedge \mathbf{R}_2)^\vee}{\boldsymbol{\tau}}, \\ \frac{\partial(\mathbf{R}_1 \circ \mathbf{R}_2)}{\partial \mathbf{R}_2} &= \mathbf{I}. \end{aligned} \quad (4.39)$$

This example implies that adjustments to the first joint  $\mathbf{R}_1$  affect the final orientation  $\mathbf{R}$  through a transformation influenced by the current state of the second joint  $\mathbf{R}_2$ . However, changes in the second joint  $\mathbf{R}_2$  directly impact  $\mathbf{R}$  without being influenced by the first joint  $\mathbf{R}_1$ .

#### 4.4 Numerical Challenges of Automatic Differentiation and Modern Libraries

AutoDiff is a cornerstone technique for computing derivatives accurately and efficiently in various optimization contexts, including differentiation on manifolds. Differentiation on manifolds poses unique challenges due to the complex geometrical properties inherent in manifold structures, which can affect the performance and applicability of AutoDiff. In differential optimization, these challenges become pronounced as AutoDiff interacts with the curved space of manifolds, potentially introducing numerical instability and inaccuracies.

This section provides examples of numerical issues that arise when using automatic differentiation for manifold-based optimization tasks. Particular attention is paid to the complexities involved in maintaining numerical stability and precision in the presence of manifold constraints, such as those found in constrained optimization and in systems defined by differential equations on manifolds. We will

take the PyPose library [1142] as an example, which defines a general data structure, `LieTensor` for Lie Group and Lie Algebra. Specifically, we will show some numerical challenges and how PyPose tackles these challenges.

**Example 4.6** (Exponential Mapping and Quaternions) The exponential map is a fundamental concept in the theory of Lie groups and is particularly critical when transitioning between Lie algebras and Lie groups represented by quaternions. This mapping enables the translation of angular velocities from the algebraic structure in  $\mathbb{R}^3$  to rotational orientations in the group of unit quaternions  $\mathbb{S}^3$ . Analytically, the exponential map for quaternions is derived from the Rodrigues' rotation formula, which relates a vector in  $\mathbb{R}^3$  to the corresponding rotation. Given a vector  $\boldsymbol{x}$  in  $\mathbb{R}^3$ , representing the axis of rotation scaled by the rotation angle, the quaternion representation of the rotation is given by:

$$\text{Exp}(\boldsymbol{\nu}) = \left[ \sin\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \frac{\boldsymbol{\nu}^\top}{\|\boldsymbol{\nu}\|}, \cos\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \right]^\top \quad (4.40)$$

where  $\|\boldsymbol{\nu}\|$  represents the magnitude of  $\boldsymbol{\nu}$ , corresponding to the angle of rotation, and  $\frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|}$  is the unit vector in the direction of  $\boldsymbol{\nu}$ .

One of the challenges of implementing a differentiable `LieTensor` is that one often need to calculate numerically problematic terms such as  $\frac{\sin \boldsymbol{\nu}}{\boldsymbol{\nu}}$  in (4.40) for the Exp and Log mapping [1080]. The direct computation of sine and cosine functions for very small angles can lead to precision issues due to the finite representation of floating-point numbers in computer systems. To manage these issues and maintain numerical stability, PyPose takes the Taylor expansion to avoid calculating the division by zero.

$$\text{Exp}(\boldsymbol{\nu}) = \begin{cases} \left[ \boldsymbol{\nu}^T \gamma_e, \cos\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \right]^T & \text{if } \|\boldsymbol{\nu}\| > \text{eps} \\ \left[ \boldsymbol{\nu}^T \gamma_o, 1 - \frac{\|\boldsymbol{\nu}\|^2}{8} + \frac{\|\boldsymbol{\nu}\|^4}{384} \right]^T & \text{otherwise,} \end{cases} \quad (4.41)$$

where  $\gamma_e = \frac{\sin(\frac{\|\boldsymbol{\nu}\|}{2})}{\|\boldsymbol{\nu}\|}$  when  $\|\boldsymbol{\nu}\|$  is significant, and  $\gamma_o = \frac{1}{2} - \frac{\|\boldsymbol{\nu}\|^2}{48} + \frac{\|\boldsymbol{\nu}\|^4}{3840}$  for small  $\|\boldsymbol{\nu}\|$ , ensuring precise calculations across all ranges of rotation magnitudes. Here,  $\text{eps}$  is the smallest machine number where  $1 + \text{eps} \neq 1$ . This analytical-to-numerical progression demonstrates the importance of accurate and stable methods for computing exponential maps in applications that require high-fidelity rotation representations.

`LieTensor` is different from the existing libraries in several aspects: (1) PyPose supports AutoDiff for any order gradient and is compatible with most popular devices, such as CPU, GPU, TPU, and Apple silicon GPU, while other libraries like LieTorch [1080] implement customized CUDA kernels and only support 1<sup>st</sup>-order gradient. (2) `LieTensor` supports parallel computing of gradient with the

vmap operator, which allows it to compute Jacobian matrices much faster. (3) Libraries such as LieTorch, JaxLie [1241], and Theseus only support Lie groups, while PyPose supports both Lie groups and Lie algebras. As a result, one can directly call the Exp and Log maps from a `LieTensor` instance, which is more flexible and user-friendly. Moreover, the gradient with respect to both types can be automatically calculated and back-propagated. The readers may find a list of supported `LieTensor` operations in [1141] and the tutorial of PyPose is available in [1144]. The usages of a `LieTensor` and its automatic differentiation can be found at <https://github.com/pypose/slambook-snippets/blob/main/lietensor.ipynb>.

#### 4.4.1 Example of Implementation of Differentiable Optimization

To enable end-to-end learning with bilevel optimization, one needs to integrate general optimizers beyond gradient-based methods such as SGD [927] and Adam [578] required by neural methods, since many problems in SLAM require other optimizations algorithms such as constrained or 2<sup>nd</sup>-order optimization [51]. Moreover, practical problems have outliers, hence one needs to robustify the loss as described in Chapter 3. Next we consider an IRLS approach to SLAM as introduced in Section 3.3, and present the intuition behind the optimization-oriented interfaces of PyPose, including `solver`, `kernel`, `corrector`, and `strategy` for using the 2<sup>nd</sup>-order Levenberg-Marquardt (LM) optimizer.

Let us start by considering a weighted least square problem:

$$\min_{\mathbf{y}} \sum_i (\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i)^T \boldsymbol{\Sigma}_i (\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i), \quad (4.42)$$

where  $\mathbf{h}(\cdot)$  is a regression model (`Module`),  $\mathbf{x} \in \mathbb{R}^n$  is the parameters to be optimized,  $\mathbf{h}_i$  denotes prediction for the  $i$ -th input sample,  $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$  is a square information matrix. The solution to (4.42) of an LM algorithm is computed by iteratively updating an estimate  $\mathbf{x}_t$  via  $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \boldsymbol{\delta}_t$ , where the update step  $\boldsymbol{\delta}_t$  is computed as:

$$\sum_i (\boldsymbol{\Lambda}_i + \lambda \cdot \text{diag}(\boldsymbol{\Lambda}_i)) \boldsymbol{\delta}_t = - \sum_i \mathbf{J}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i, \quad (4.43)$$

where  $\mathbf{r}_i = \mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i$  is the  $i$ -th residual error,  $\mathbf{J}_i$  is the Jacobian of  $\mathbf{h}$  computed at  $\mathbf{x}_{t-1}$ ,  $\boldsymbol{\Lambda}_i$  is an approximated Hessian matrix computed as  $\boldsymbol{\Lambda}_i = \mathbf{J}_i^T \boldsymbol{\Sigma}_i \mathbf{J}_i$ , and  $\lambda$  is a damping factor. To find step  $\boldsymbol{\delta}_t$ , one needs a linear `solver`:

$$\mathbf{A} \cdot \boldsymbol{\delta}_t = \boldsymbol{\beta}, \quad (4.44)$$

where  $\mathbf{A} = \sum_i (\boldsymbol{\Lambda}_i + \lambda \cdot \text{diag}(\boldsymbol{\Lambda}_i))$ ,  $\boldsymbol{\beta} = - \sum_i \mathbf{J}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i$ . In practice, the square matrix  $\mathbf{A}$  is often positive-definite, so we could leverage standard linear solvers such as Cholesky. If the Jacobian  $\mathbf{J}_i$  is large and sparse, we may also use sparse solvers such as sparse Cholesky [161] or preconditioned conjugate gradient (PCG) [459] solver.

In practice, one often introduces robust `kernel` functions  $\rho : \mathbb{R} \mapsto \mathbb{R}$  into (4.42) to reduce the effect of outliers:

$$\min_{\mathbf{y}} \sum_i \rho(\mathbf{r}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i), \quad (4.45)$$

where  $\rho$  is designed to down-weigh measurements with large residuals  $\mathbf{r}_i$ . In this case, we need to adjust (4.43) to account for the presence of the robust kernel. A popular way is to use an IRLS method, Triggs' correction [1104], which is also adopted by the Ceres [15] library. However, it needs 2<sup>nd</sup>-order derivative of the kernel function  $\rho$ , which is always negative. This can lead 2<sup>nd</sup>-order optimizers including LM to become unstable [1104]. Alternatively, PyPose introduces an IRLS method, `FastTriggs`, which is faster yet more stable than `Triggs` by only involving the 1<sup>st</sup>-order derivative:

$$\mathbf{r}_i^\rho = \sqrt{\rho'(c_i)} \mathbf{r}_i, \quad \mathbf{J}_i^\rho = \sqrt{\rho'(c_i)} \mathbf{J}_i, \quad (4.46)$$

where  $c_i = \mathbf{r}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i$ ,  $\mathbf{r}_i^\rho$  and  $\mathbf{J}_i^\rho$  are the corrected model residual and Jacobian due to the introduction of kernel functions, respectively. More details about `FastTriggs` and its proof can be found in [1140], while IRLS was introduced in Section 3.3.

A simple LM optimizer may not converge to the global optimum if the initial guess is too far from the optimum. For this reason, we often need other strategies such as adaptive damping, dogleg, and trust region methods [694] to restrict each step, preventing it from stepping “too far”. To adopt those strategies, one may simply pass a `strategy` instance, *e.g.*, `TrustRegion`, to an optimizer. In summary, PyPose supports easy extensions for the aforementioned algorithms by simply passing `optimizer` arguments to their constructor, including `solver`, `strategy`, `kernel`, and `corrector`. A list of available algorithms and examples can be found in [1143]. The usages of a 2nd-order optimization can be found at <https://github.com/pypose/slambook-snippets/blob/main/optimization.ipynb>.

#### 4.4.2 Related Open-source Libraries

Open-source libraries related to differentiable optimization can be divided into three groups: (1) linear algebra, (2) machine learning libraries, and (3) specialized optimization libraries.

**Linear Algebra Libraries** are essential to machine learning and robotics research. NumPy [820], a linear algebra library for Python, offers comprehensive operations on vectors and matrices while enjoying higher running speed due to its underlying well-optimized C code. Eigen [412], a high performance C++ linear algebra library, has been used in many projects such as TensorFlow [2], Ceres [15], GTSAM [252], and g<sup>2</sup>o [402]. ArrayFire [724], a GPU acceleration library for C, C++, Fortran, and Python, contains simple APIs and provides GPU-tuned functions.

**Machine Learning Libraries** focus more on operations on tensors (*i.e.*, high-dimensional matrices) and automatic differentiation. Early machine learning frameworks, such as Torch [228], OpenNN [827], and MATLAB [735], provide primitive tools for researchers to develop neural networks. However, they only support CPU computation and lack concise APIs, which limit their usability. A few years later, deep learning frameworks such as Chainer [1098], Theano [24], and Caffe [517] arose to handle the increasing size and complexity of neural networks while supporting multi-GPU training with convenient APIs for users to build and train their neural networks. Furthermore, the recent frameworks, such as TensorFlow [2], PyTorch [847], and MXNet [184], provide a comprehensive and flexible ecosystem (*e.g.*, APIs for multiple programming languages, distributed data parallel training, and facilitating tools for benchmark and deployment). Gvnn [430] introduced differentiable transformation layers into Torch-based framework, leading to end-to-end geometric learning. JAX [110] can automatically differentiate native Python and NumPy functions and is an extensible system for composable function transformations. In many ways, the existence of these frameworks facilitated and promoted the growth of deep learning. Recently, more efforts have been taken to combine standard optimization tools with deep learning. Recent work like Theseus [870] and CvxpypLayer [18] showed how to embed differentiable optimization within deep neural networks. PyPose [1142] incorporates 2<sup>nd</sup>-order optimizers such as Gaussian-Newton and Levenberg-Marquardt and can compute any order gradients of Lie groups and Lie algebras, which are essential to robotics.

**Other Specialized Optimization Libraries** have been developed and leveraged in robotics. To mention a few, Ceres [15] is an open-source C++ library for large-scale nonlinear least squares optimization problems and has been widely used in SLAM. Pyomo [434] and JuMP [292] are optimization frameworks that have been widely used due to their flexibility in supporting a diverse set of tools for constructing, solving, and analyzing optimization models. CasADi [32] has been used to solve many real-world control problems in robotics due to its fast and effective implementations of different numerical methods for optimal control. Pose-and factor-graph optimization also play an important role in robotics. For example, g<sup>2</sup>o [402] and GTSAM [252] are open-source C++ frameworks for graph-based nonlinear optimization, which provide concise APIs for constructing new problems and have been leveraged to solve several optimization problems in SLAM.

Optimization libraries have also been widely used in robotic control problems. To name a few, IPOPT [1136] is an open-source C++ solver for nonlinear programming problems based on interior-point methods and is widely used in robotics and control. Similarly, OpenOCL [589] supports a large class of optimization problems such as continuous time, discrete time, constrained, unconstrained, multi-phase, and trajectory optimization problems for real-time model-predictive control. Another library for large-scale optimal control and estimation problems is CT [383], which provides standard interfaces for different optimal control solvers and can be

extended to a broad class of dynamical systems in robotic applications. Drake [1076] has solvers for common control problems and that can be directly integrated with its simulation tool boxes. Its system completeness made it favorable to researchers.

#### 4.5 Further Readings & Recent Trends

Deep learning methods have witnessed significant development in recent years [1283]. As data-driven approaches, they are believed to perform better on visual tracking compared to traditional handcrafted features. Most studies on the subject employ end-to-end structures, including both supervised methods such DeepVO [1160] and TartanVO [1164] and unsupervised methods such as UnDeepVO [654] and Unsupervised VIO [1170]. It is generally observed that the supervised approaches achieve higher performance compared to their unsupervised counterparts since they can learn from a diverse range of ground truths such as pose, flow, and depth. Nevertheless, obtaining such ground truths in the real world is a labor-intensive process [1163].

Recently, hybrid methods have received increasing attention as they integrate the strengths of both geometry-based and deep-learning approaches. Several studies have explored the potential of integrating BA with deep learning methods to impose topological consistency between frames, such as attaching a BA layer to a learning network such as BA-Net [1065] and DROID-SLAM [1079]. Additionally, some works focused on compressing image features into codes (embedded features) and optimizing the pose-code graph during inference such as DeepFactors [236]. Furthermore, DiffPoseNet [839] is proposed to predict poses and normal flows using networks and fine-tune the coarse predictions through a Cheirality layer. However, in these works, the learning-based methods and geometry-based optimization are decoupled and separately used in different sub-modules. The lack of integration between the front-end and back-end may result in sub-optimal performance. Besides, they only back-propagate the pose error “through” BA, thus the supervision is from the ground truth poses. In this case, BA is just a special layer of the network. Recently, iSLAM [346] connects the front-end and back-end bidirectionally and enforces the learning model to learn from geometric optimization through a bilevel optimization framework, which achieves performance improvement without external supervision. Some other tasks can also be formulated as bilevel optimization, *e.g.*, reinforcement learning [1024], local planning [1213], global planning [186], feature matching [1264], and multi-robot routing [419].

# 5

## Dense Map Representations

Victor Reijgwart, Jens Behley, Teresa Vidal-Calleja, Helen Oleynikova,  
Lionel Ott, Cyrill Stachniss and Ayoung Kim

We now shift our focus to a different aspect of SLAM, specifically its mapping component. The mapping problem is approached with the assumption that the robot’s pose has been estimated, and the objective is to construct a dense map of its surroundings. Indeed, typical approaches first solve for the robot trajectory using the SLAM back-end —as discussed in the previous chapters— and then reconstruct a dense map given the trajectory. In this chapter, we illustrate the details of the dense map representation, focusing on the map elements, data structures, and methods.

Early mapping approaches were predominantly based on sparse, landmark-based solutions as discussed in Chapter 1, which extract only a few salient features from the environment. However, the increase in compute capabilities paired with the advent of accurate 3D range sensors, such as mechanical 3D LiDARs or RGB-D cameras that provide detailed 3D range measurements at high frequencies, led to an increasing research interest in dense map representations. Dense maps are crucial for downstream tasks that require a detailed understanding of the environment, such as planning, navigation, manipulation and precise localization. This chapter explains how these dense methods leverage the full spectrum of range sensor data to refine and update comprehensive maps.

The chapter begins by presenting key sensor types that facilitate dense mapping, primarily focusing on range sensors that produce detailed range measurements, which are reviewed in Section 5.1. The chapter continues with an introduction to fundamental representation elements and data structures in Section 5.2, that we then tailor to specific applications in Section 5.4. Contrasting with sparse landmark-based mapping, the choice of a dense map representation hinges on the sensor types used and the intended downstream applications. Key factors influencing the selection of the representation type are summarized in Section 5.5.

### 5.1 Range Sensing Preliminaries

*range sensor*

Before we delve into dense map representations, we briefly summarize a key sensing modality, range sensors, often used for SLAM, providing the necessary context for the following discussion. Such sensors produce range measurements to the objects

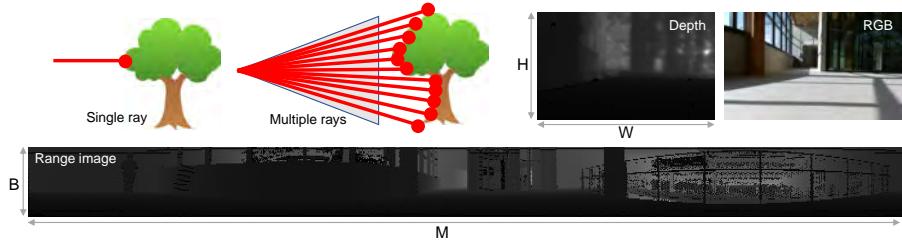


Figure 5.1 Single ray and multi-ray types for LiDAR sensors. Sample data from real-world is visualized to show RGB, depth, and LiDAR range image.

in the environment, including LiDAR sensors, time-of-flight (TOF) cameras, RGB-D cameras, and stereo cameras. Here, we concentrate on the most commonly used LiDAR sensors and RGB-D cameras that are predominately used in outdoor and indoor environments for SLAM and dense mapping.

### 5.1.1 Sensor Measurement Model

Let us start with a brief summary of the sensing mechanism and associated measurement model. In the case of LiDAR sensors,<sup>1</sup> the range measurements are generated using laser beams that are emitted, reflected by the environment, and then detected [932]. By measuring the time  $t_{\text{emit}}$  when the laser beam is emitted and the time of detection  $t_{\text{detect}}$ , we can derive the range  $r$  using the speed of light  $c$  as follows:

$$r = \frac{c(t_{\text{detect}} - t_{\text{emit}})}{2}. \quad (5.1)$$

A single ray measurement can be enhanced into a two-dimensional (2D) or three-dimensional (3D) collection of points by employing an array of rays that move in a designated pattern, such as a 360-degree rotation or a specific shape. The collection of points generated by the sensor is referred to as point clouds, which serve as the fundamental element for creating maps. The LiDAR measurements can also be represented using a range image  $\mathbf{R} \in \mathbb{R}^{B \times M}$ , where the range of each of the  $B$  beams is stored for a single complete turn of the sensor, *i.e.*, a complete 360° rotation. Thus, we have  $M$  measurements in the horizontal field of view of the sensor for each of the  $B$  beams; see illustrations and examples in Figure 5.1.

Other commonly used range sensors in robotics applications are RGB-D cameras, such as Microsoft’s Kinect and Azure Kinect DK, and Intel’s RealSense. RGB-D cameras provide—besides the RGB image  $\mathbf{I}_{\text{RGB}} \in \mathbb{R}^{3 \times H \times W}$  of height  $H$  and width  $W$ —a depth map  $\mathbf{I}_D \in \mathbb{R}^{H \times W}$  of the same dimension, where each pixel location contains the depth or range. To generate the depth map  $\mathbf{I}_D$ , early RGB-D cameras

<sup>1</sup> We illustrate this chapter using simple 2D and mechanically rotating 3D LiDARs, while other solid-state and flash LiDARs will be introduced in Chapter 8

use a structured infrared (IR) light source with a known pattern that is projected onto the environment. The distance of individual pixels is then determined from the distortion of the known pattern. As sunlight usually interferes with this sensing mechanism, such RGB-D cameras using projected light are mainly used in indoor environments. Fortunately, newer generations of RGB-D sensors introduce an IR texture projector or time-of-flight (TOF), which are less affected by interferences, supporting outdoor applications.

### 5.1.2 Conversion to Point Cloud

Using the intrinsics of a range sensor (*e.g.*, a LiDAR or RGB-D camera), we can convert a range image  $\mathbf{R}$  or a depth map  $\mathbf{I}_D$  into a point cloud  $\mathbf{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_N\}$ , where points  $\mathbf{p}_i \in \mathbb{R}^3$  are expressed in the local coordinate frame of the sensor. The point is the most fundamental unit in the map representation and will be discussed further in this chapter.

For conversion from LiDAR, the sensors provide an intrinsic calibration for each beam  $(\phi_{i,j}, \theta_{i,j})$ , where  $1 \leq i < B$  and  $1 \leq j < M$ , consisting of the azimuthal angle  $\phi_{i,j} \in [0, 2\pi]$  and polar/inclination angle  $\theta_{i,j} \in [-\pi, \pi]$  as depicted in Figure 5.2. Using these known angles of each beam, we can convert a range measurement  $r_{i,j}$  at  $\mathbf{R}_{i,j}$  into a three-dimensional point  $\mathbf{p} = (x, y, z)$  as follows:

$$x = r_{i,j} \cos(\theta_{i,j}) \cos(\phi_{i,j}) \quad (5.2)$$

$$y = r_{i,j} \cos(\theta_{i,j}) \sin(\phi_{i,j}) \quad (5.3)$$

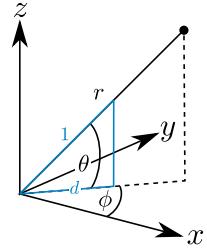
$$z = r_{i,j} \sin(\theta_{i,j}). \quad (5.4)$$

For an RGB-D camera, we commonly use a pinhole camera model to convert the ranges  $r_{u,v} = \mathbf{R}_{u,v}$  at pixel location  $(u, v)$  into a three-dimensional coordinate. For this, we use the intrinsics of the camera  $\mathbf{K} \in \mathbb{R}^{3 \times 4}$  to convert a homogeneous coordinate  $\mathbf{x} = (u, v, r_{u,v})$  in the image coordinate into a point  $\mathbf{p}$  in the camera coordinate while  $\dagger$  being pseudo inverse:

$$\mathbf{p} = \mathbf{K}^\dagger \mathbf{x}. \quad (5.5)$$

The resulting point cloud is said to be *organized* or *unorganized* depending on how the points are structured. When converted from a depth map, the point cloud is organized, and each point location is structured with respect to the pixel location of the associate depth map. This can be exploited to compute neighboring points by a simple indexing. On the other hand, the point cloud generated by a LiDAR sensor is more complicated. For example, the generated point cloud is organized in the static 3D mechanical LiDAR. However, the organization of the point cloud no

Figure 5.2 Spherical coordinate



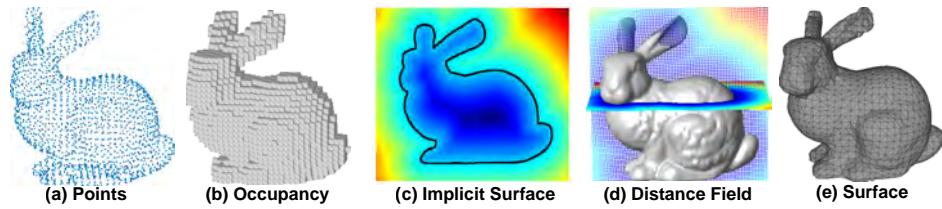


Figure 5.3 Examples of common dense representations.

longer holds in the case of non-repeated pattern solid-state, flash LiDARs, or mechanically rotating LiDAR under motion distortion. In many cases, the unorganized points are distorted due to the movement of the sensor and measurement neighborhood in the range image is not necessarily correlated to spatial neighborhood. Therefore, an estimate of the motion of the sensor while completing a sweep, *e.g.*, using IMU measurements or odometry information, together with the information of the per-beam time is necessary to undistort a LiDAR point cloud to account for the motion of the sensor [1130, 256, 1228]. For more details on motion distortion and compensation, refer to Chapter 8.

## 5.2 Foundations of Dense Mapping

A map, generated with sensors' information and data processing approaches, is a symbolic structure that models the environment [1088, 136]. One thing to be noted here is that the map representation can be diverse, and many different representations exist for the same spatial information (as in Figure 5.3). The choice and accuracy of the scene representation strongly impact the performance of the task at hand, and thus the representation should be determined by the use case. For instance, motion estimation and localization in robotics favor sparse representation, such as 3D points features [784, 893] in order to exploit these features for consistent robot pose estimation. On the other hand, a key objective of scene reconstruction is an accurate, dense, and high-resolution map, for example, for inspection purposes [506, 819, 919]. Similarly, path planning tasks require dense information such as obstacle occupancy or closest distance to collision for obstacles avoidance [771, 304, 1107]. Overall, this chapter examines the following three questions, and is organized into sections that address each in turn.

**Q1. What quantity do we need to estimate for dense mapping?** The most commonly used quantities to represent the environment are *occupancy* and *distance*. Occupancy is a key property in mapping for distinguishing between free and occupied space. Distance estimation provides a more robot-centric interpretation of free and occupied space by measuring the range to nearby surfaces or objects.

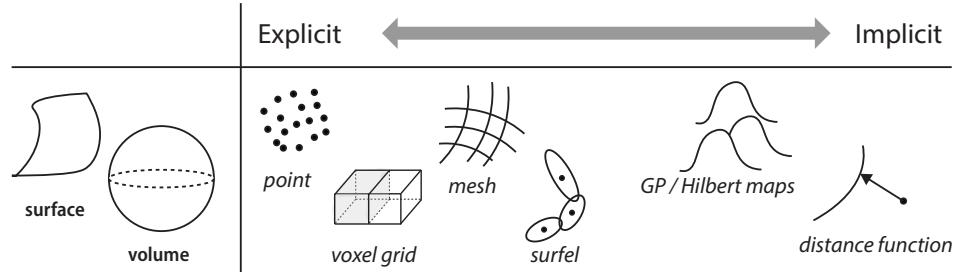


Figure 5.4 The representation can be either explicit or implicit. The illustration is simplified for clarity. In reality, the abstraction is not clearly separable and can often be applied in a combined manner.

**Q2. How should the world be represented?** This is the question of what space abstraction we use for representation. Broadly speaking, a representation can be either explicit or implicit. Explicit space abstractions are further classified based on the type of geometry they utilize, while implicit representation can be categorized based on their choice of functions. The list of abstraction types are illustrated in Figure 5.4.

**Q3. What data structure and storage should be used?** The chosen representation should be stored in memory for later use. The data structure and storage method should be selected based on the specific application and intended usage.

In the literature, a wide variety of approaches exist for generating a dense representation of the scene using range sensors, varying in terms of their estimated quantities, space abstraction, storage structure, continuity, and application areas.

Beginning the discussion on different representations, we first explore the primary quantities estimated from range measurements. The focus is on understanding the key quantities predominantly estimated in the mapping phase. We will provide a concise overview of the basic definitions of each quantity, elaborating their significance and the specific contexts in which they play a crucial role.

### 5.2.1 Occupancy Maps

Since their introduction over three decades ago by Elfes and Moravec [304, 771], occupancy grids have been widely used. Their simplicity and computational efficiency have made occupancy grids<sup>2</sup> popular when mapping indoor (and even outdoor) environments. In the simplest scenario, the estimated quantity is the *probability* of a cell being occupied. In this case, the occupancy of the cell is modeled as a probability of that cell containing an obstacle, with occupancy equal to 1 for occu-

<sup>2</sup> Occupancy mapping can be conducted without grid-based methods (*e.g.*, GPOM [813]). In this chapter, we will focus on grid-based mapping for simplicity.

pied cells and 0 for cells deemed empty. Essentially occupancy mapping is a binary classification problem to predict the binary class probability of each cell.

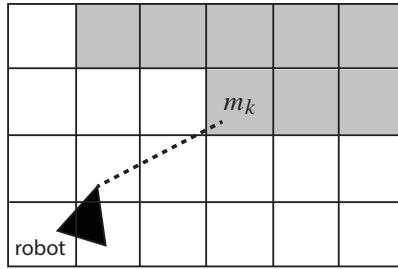


Figure 5.5 A simple ray-casting example in 2D. Given a range measurement at a certain (computed or estimated) azimuth, the return of the range measurement indicates the existence of an obstacle.

Given a set of sensor measurements  $\mathbf{z}_{1:t}$  and a set of sensor poses  $\mathbf{x}_{1:t}$ , the probability of being occupied for each cell in the map  $\mathbf{m}$  is modeled as  $p(\mathbf{m}|\mathbf{z}_{1:t}, \mathbf{x}_{1:t})$ . A sample map is shown in Figure 5.5. Assuming that each cell  $m_k$  is independent and that the measurements are conditionally independent, the update of occupancy can be formulated efficiently using the well-known log-odds form [771],

$$l(m_k|\mathbf{z}_{1:t}) = l(m_k|\mathbf{z}_{1:t-1}) + l(m_k|\mathbf{z}_t), \quad (5.6)$$

where  $l(\cdot|\cdot) = \log(o(\cdot|\cdot))$ , and  $o(\cdot|\cdot)$  is the odds form:

$$o(m_k|\mathbf{z}_{1:t}) = \frac{p(m_k|\mathbf{z}_{1:t})}{1 - p(m_k|\mathbf{z}_{1:t})}. \quad (5.7)$$

The main advantage of occupancy mapping is shown in (5.6), where only the previous occupancy value and the inverse sensor model  $l(m_k|\mathbf{z}_t)$  are needed to update the probability through a simple addition. Despite these benefits, occupancy grids rely on very strong assumptions of the environment to be efficient. Notably, the assumption that the likelihood of occupancy in one cell is independent of other cells disregards spatial correlations, which can be important to infer occupancy in unobserved nearby regions. Additionally, traditional occupancy grids require the discretization of the environment be defined a priori, which makes the spatial resolution constant throughout the map.

### 5.2.2 Implicit Surface and Distance Fields

Another way of representing the geometry of the robot's surrounding is not through *probabilities* of occupancy, but rather by describing the boundary between free and occupied space. In an ideal world, we could describe the shape and location of this

boundary using an analytical function in three variables ( $x$ ,  $y$  and  $z$ ) which reaches 0 whenever a point  $\mathbf{p} = (x, y, z)$  lies on the surface. In other words, the function's zero crossings correspond to the surface itself. Such a function is known as an *implicit surface*. By convention, the function's sign is negative when  $\mathbf{p}$  lies *inside* an object and positive *outside*. A simple example in Figure 5.6 illustrates how this implicit function values are determined.

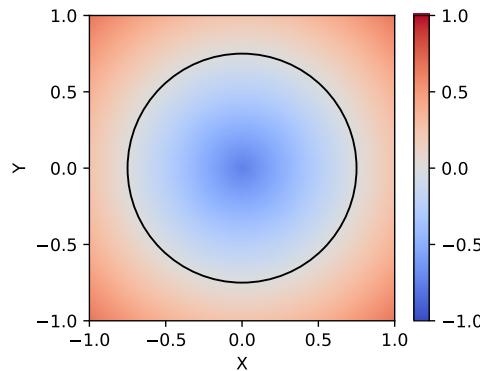


Figure 5.6 A solid 2D disk represented as an explicit surface (black outline) and implicit surface (colored field). The implicit surface function is negative inside the object (blue) and positive outside (red). Note how the function's zero crossings correspond to the surface itself.

There are many advantages to continuous implicit surface representations. Given that there is no fixed resolution, they can represent objects of arbitrary shapes at any level of detail. Furthermore, they make it possible to check if a given point in space is inside or outside an obstacle by simply evaluating the function's sign.

Different types of functions can be employed to model implicit surfaces, with the Euclidean Signed Distance Function (ESDF) being a prevalent option. At any query point, the ESDF expresses the distance to the nearest surface (indicated by the magnitude of the ESDF) and whether the point is inside an obstacle (indicated by the sign of the ESDF). ESDF representations are commonly used by accelerated geometric algorithms for tasks such as collision checking. Furthermore, high-quality proximity gradients can be derived from the ESDF for optimization-based motion planning and shape registration.

The ESDF is computed by finding the (Euclidean) closest surface point for each point in the map. For a known surface, this can efficiently be done using techniques such as Fast Marching. However, for surface estimation, the *projective* signed distance is more commonly used as it can efficiently be computed from measurements and is better suited for filtering. Given a measurement ray going through a query point, the projective distance is defined as the distance from the beam's endpoint to the query point *along the ray*. This eliminates the need to search the closest point

explicitly. Although the projective distance overestimates the Euclidean distance, its zero crossings (the estimated surface) remain correct. The standard approach of estimating implicit surfaces, proposed by Curless and Levoy [234] and popularized in the field of robotics by KinectFusion [798], combines the projective signed distances for all measurements using a simple weighted average. To reduce the impact of overestimates, the projective signed distance function is typically clamped to a fixed range, named the truncation band, in which case it is called the Truncated Signed Distance Function (TSDF). Note that ESDFs can efficiently be computed from TSDFs and occupancy maps, as will be described in Section 5.4.4.

*Truncated Signed Distance Function*

### 5.2.3 Occupancy Maps or Distance Fields?

Being volumetric methods, a shared aspect of these estimated quantities in occupancy and implicit representations is that they model the geometry by estimating a quantity of interest everywhere in the observed volume. However, each representation fundamentally prioritizes different things. Which option is best, therefore, depends on the application. We will briefly summarize two key differences.

**Directness in modeling:** Given that measurement rays *directly* tell us which parts of space are free, occupied, or unobserved, maps based on occupancy probabilities can be updated using fewer heuristics and assumptions. In contrast, implicit surfaces typically model the distance to the surface. This can be computed exactly for a known surface, but not from partial measurements. For surface estimation, they therefore rely on distance proxies such as the previously introduced TSDF.

**Smoothness:** Implicit surface maps are inherently smoother than occupancy maps, which model a binary property. The smoothness of implicit surfaces has many benefits. Most importantly, it makes them differentiable. The resulting proximity gradients are valuable for many applications. Smoothness also reduces the approximation errors resulting from discretization and makes it possible to obtain good, sub-pixel resolution estimates through interpolation. However, since discontinuities cannot be represented smoothly, implicit surfaces tend to miss thin obstacles.

## 5.3 Map Representations

### 5.3.1 Explicitness of Target Spatial Structures

As summarized in Figure 5.4, the representation can be classified based on their explicitness and target space. In 3D mapping, representing volume is straightforward; however, surfaces are equally important in robotic mapping for enabling downstream tasks. We can consider four major categorization: explicit surface, implicit surface, explicit volume, and implicit volume representations.

For surfaces, we can either *explicitly* or *implicitly* represent a surface. Defined as *surface* a 2D manifold, an explicit type of representation aims to characterize the space

in terms of their boundary of the objects in the scene. The simplest abstraction that can represent the boundary is directly the point cloud produced by the range sensors. Another general representation of the surface is the polygon mesh (Section 5.4.3), which comprises vertices, edges, and faces. Meshes have the ability to encode the directed surfaces of a volume by forming connected closed polygons, more commonly triangles. Surfels (Section 5.4.2) are also popular abstractions and are widely used in mapping. Surface representations are a key for any visualization application, but also are used for rendering simulated environments, augmented reality or for Computer-aided design and 3D printing.

Similar strategies are employed in 3D volume modeling. Naive point-based representations are commonly used in LiDAR SLAM. Additionally, occupancy or distance-based voxels (Section 5.4.4) are popular choices for explicit representation. When storing volumetric maps, careful consideration of data storage is necessary to minimize computational costs. Implicit representations for volumetric mapping are also utilized, typically through functions. Gaussian process (GP) (Section 5.4.5) and Hilbert maps (Section 5.4.6) are well-known examples of implicit representations.

### **5.3.2 Types of Spatial Abstractions**

#### *5.3.2.1 Points*

*points*

Given range measurements, a straightforward dense map representation is using point clouds. For generation, we accumulate the point clouds  $P_t$  recorded at time  $t$  in the local coordinate frame  $\mathcal{F}^t$  using the estimated global pose  $T_t^1$  in a global map point cloud  $P_M$ . Also common practice is to assume our global coordinate frame of  $P_M$  is given in the coordinate frame of the first point cloud  $\mathcal{F}^1$ .

Since simply accumulating point clouds  $P_1, \dots, P_t$  does not scale to larger environments, a common strategy is to discard redundant measurements of the same spatial location. To this end, most methods [1266, 256, 1130] use efficient nearest neighbor search, such as voxel grids or hierarchical tree-based representations (see Section 5.3.3), to subsample and store the point clouds, *e.g.*, store only a limited number of points per voxel [256, 1130] or only specific points that meet a certain criterion are stored [1266]. Additionally, a representation of only keyframes where only a few point clouds are explicitly stored is possible, but this requires to determine when a keyframe or submap needs to be generated.

Being the most elemental representation form, a point cloud map can be converted into other representations, *e.g.*, a mesh via Poisson surface reconstruction [1128] or a Signed Distance Function (SDF) via marching cubes [1129]. Unfortunately, this is feasible only with additional data, such as the viewpoint of a point's measurement. Yet, this information may be lost when merging multiple measurements into a point cloud map. Therefore, assumptions about the surface's direction are often required to discern inside or outside regions.

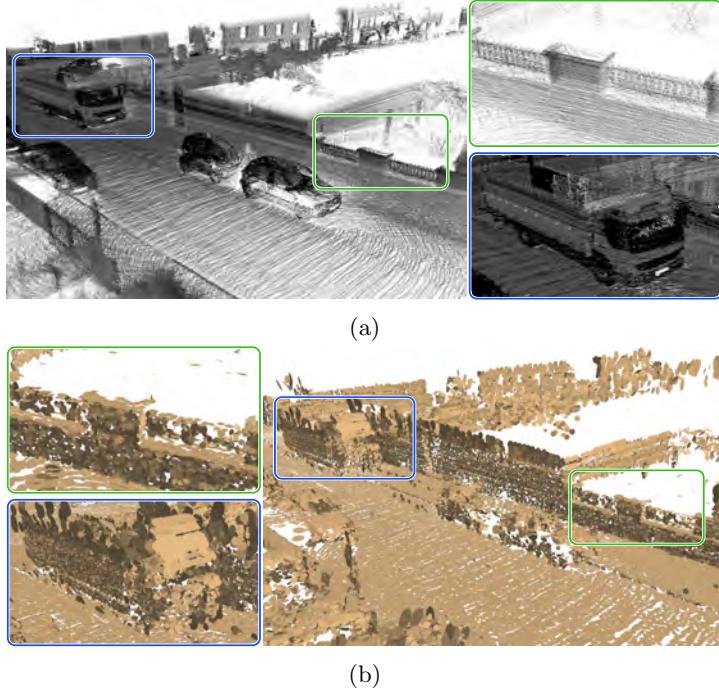


Figure 5.7 Qualitative comparison of maps generated by accumulating point clouds and surfels from a sequence of LiDAR scans from the KITTI dataset [373] Sequence 07. (a) Point cloud map. The brightness of points indicates the remission of the LiDAR measurements. (b) Corresponding surfel map based on circular disks. The complete map with all accumulated point clouds use 2.95 GB, while the corresponding surfel map by SuMa [66] uses only 160 MB.

### 5.3.2.2 Surfels

While point cloud maps directly represent the measurements, the stored points do not contain surface information or can represent from which direction a point has been measured. With surfels [867], we can encode such information by adding directional information to a point. Surfels are commonly represented via circular or elliptic discs [548, 66, 1181, 104, 103], or more generally ellipsoids [1038, 1039, 282, 1315] modeled with a Gaussian. *Splatting* [1315, 103] allows integrating texture information but also blending overlapping surfels into coherent renderings of a specific viewpoint; see Chapter 14 for a more extensive discussion.

A commonly employed circular surfel representing a circular disk is defined by a location  $\mathbf{p} \in \mathbb{R}^3$ , a normal direction  $\mathbf{n} \in \mathbb{R}^3$ , and a radius  $r \in \mathbb{R}$ . These geometric primitives can be efficiently rendered using the capabilities of modern graphics processing units (GPUs). This accelerates point-to-surfel associations and leads to a substantial memory reduction.

*surfels*

Figure 5.7 qualitatively compares a point cloud-based and a surfel-based map representation. A dense point cloud can accurately represent the environment with a high level of detail, but at the cost of memory. In contrast, while losing fine details as multiple measurements get aggregated into a single surfel, significant memory usage can be reduced while preserving the main structural details of larger surfaces.

Closely related to the explicit geometric representation of surfaces via surfels, *i.e.*, small circular surface patches, is the representation via a normal distributions transform (NDT) [82, 1032]. Using NDT, the space is subdivided into voxels and the points inside a voxel are approximated via a normal distribution  $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , having estimated mean  $\boldsymbol{\mu}$  and covariance from the enclosed points  $\boldsymbol{\Sigma}$ . The eigenvalues  $\lambda_1 < \lambda_2 < \lambda_3$  and corresponding eigenvectors  $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$  of the covariance can be used to estimate the surface properties inside a voxel. For planar surfaces ( $\lambda_1 \ll \lambda_2$ ), the eigenvector  $\mathbf{v}_1$  of the smallest eigenvalue  $\lambda_1$  corresponds to the surface normal. Thus, for planar surfaces the NDT represents a surfel, and can also more accurately represent point distributions that cannot be approximated via a surfel. In that sense, the NDT is a hybrid representation that is explicit due to the space division into a voxel grid, but also implicit due to the representation of voxels via a normal distribution which continuously represents the space inside a voxel.

*meshes*

### 5.3.2.3 Meshes

While describing local surface properties, both point clouds and surfel maps are still relatively sparse as they do not model the surface’s connectivity. One way to get a more complete understanding is to use meshes, which describe the surface as a set of points that are connected to form a collection of polygons. This, in turn, makes it possible to represent watertight surfaces, query and interpolate new surface points, and efficiently iterate along a connected surface.

In meshing terminology, each polygon is referred to as a *face*, and each corner point as a *vertex*. The most common types are triangle meshes, where each face is bounded by three vertices, and quad meshes, whose faces are bound by four vertices. Note that a polygon, or face, can always be broken down into an equivalent set of triangles; hence, triangle meshes are not only the simplest but also the most general.

A mesh is a very flexible and memory-efficient representation because the number of faces and vertices can be directly adapted to the surface complexity and required level of detail. For example, a plane of any size can be represented with just two triangular faces and four vertices. Furthermore, meshes are well-suited for parallel processing and rendering. Meshes are often used in applications that overlap with computer graphics, such as rendering, surface analysis, manipulation, and deformation, and more generally in applications involving digital models, simulation, or surface-based algorithms, such as path planning for ground robots.

### 5.3.2.4 Voxels

Point clouds and meshes are well suited to represent properties of the environment that are defined along surfaces. However, certain estimated quantities, including occupancy and Signed Distance, are defined throughout the entire volume. One straightforward way to store and process volumetric properties is to discretize them over a regular grid. Discretized occupancy and Signed Distance maps are called occupancy grids and Signed Distance Fields, respectively.

Generalizing the concept of 2D pixels, the cells in a 3D grid are referred to as voxels. Given a grid's regular structure, each voxel can easily be assigned a unique index and stored in a data structure. Note that a voxel is merely a container or, more formally, a space partition. The significance of its contents varies from one method to another. In a classic occupancy grid, a voxel's value represents the likelihood that any point in the voxel is occupied. Hence, the occupancy at an arbitrary point in the map is equal to the value of the voxel that contains the point. However, a voxel's value does not have to represent a constant little cube. For example, voxels in a Signed Distance Field estimate the signed distance at each voxel's center. To retrieve the signed distance at an arbitrary point, one would therefore query the voxels that neighbor the point and obtain the point's value using interpolation. Finally, some applications even use (sparse) voxel grids to store and efficiently query non-volumetric properties, such as points or surface colors.

### 5.3.2.5 Continuous Functions

Functions are a key abstraction for mapping in a continuous manner. The problem of mapping in this case is reduced to fitting a parametric or non-parametric function, *i.e.*, solving a regression problem. Most of the above-mentioned space abstractions require the discretization of the environment to be defined *a priori*, which usually makes the spatial resolution constant throughout the map. Continuous functions, however, parametric or non-parametric, give more flexibility allowing the resolution to be recomputed and also provide interpolation capabilities to fill up data gaps.

Some parametric functions such as infinite lines in 2D [1103] and planes in 3D [532, 374, 1103] require making strict assumptions about the environment and limit the representation of the scene. However, these representations are efficient in terms of memory consumption and computational complexity. Control points-based functions (*e.g.*, B-splines [929]) or non-parametric (*e.g.*, GP-based) have the ability to model the environment with fewer assumptions, still in a continuous manner. From occupancy [813], implicit surface [1183, 634], distance fields [1192], and surface itself [1119], GP-based representations are a popular choice to represent the environment —despite their high computational complexity— because of their probabilistic nature, which enables uncertainty quantification and inference over both observed and unseen areas [378].

A key advantage of the continuous functions for mapping is that if they are chosen

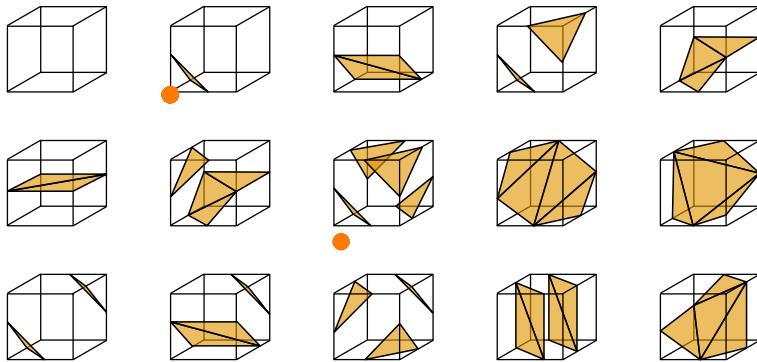


Figure 5.8 The Marching Cubes algorithm works by projecting the cubes into the implicit surface, querying the sign of the values at the corners of the cubes, and looking up which one of the 15 configurations these values map to. (Image credit: Ryoshozu, “*Marching cubes*”, licensed under CC BY 4.0. Source: <https://commons.wikimedia.org/wiki/File:MarchingCubesEdit.svg>)

to be at least once differentiable they will be able to provide gradients. Gradient information can be key for localisation to compute surface normals [1193], loop closure to compute terrain features [377], for data fusion [634] and planning [1193] applications.

#### 5.3.2.6 Conversions

The abstractions listed above are not always used exclusively; they are often converted from one form to another or employed simultaneously in multiple forms.

For instance, explicit geometry, such as points and meshes, can be transformed into an implicit surface. One flexible method to compute the signed distance at any point in space is through a *closest point lookup*, which can be performed against any explicit geometry and on-demand, only when and where needed. Alternatively, the signed distance can be computed across all points on a regular grid using *wavefront propagation*, which can efficiently be implemented via the fast marching method [988].

Conversely, it is common to convert implicit surfaces into mesh representations. The original technique for converting distance fields into meshes is known as Marching Cubes [693]. The algorithm divides the implicit surface into a grid of fixed-size *cubes*, which it processes independently. Each cube generates a set of triangle elements based on the implicit surface’s values at its eight corners (Figure 5.8). The positions of their vertices are then refined through linear interpolation. Meshes, including those from BIM (Building Information Modeling) and CAD (Computer-Aided Design) models, can, in turn, be sampled to create points or surfels.

Occasionally, a discrete representation must be converted into a continuous one.

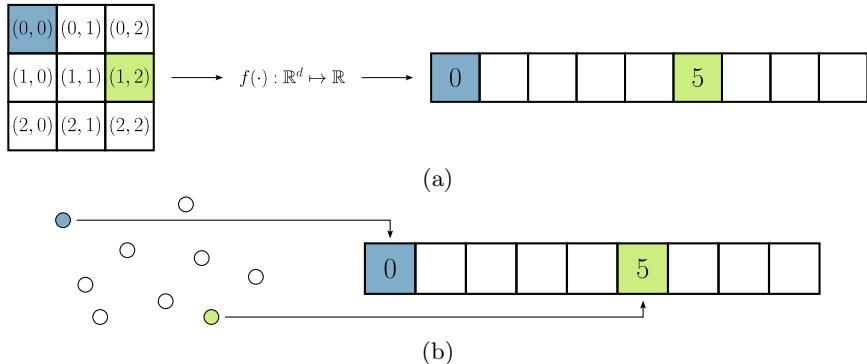


Figure 5.9 (a) Mapping of logical grid coordinates to an underlying naive array-based storage through a function  $f(\cdot)$  that uniquely maps coordinates to linear indices. (b) Storage of unordered data directly into an array structure.

This is typically achieved by solving an optimization problem over the parameters of the continuous representation, minimizing the fitting error with respect to the discrete data.

### 5.3.3 Data Structures and Storage

The abstractions introduced in the previous sections all need to be stored in memory. In this section, we explore how various abstractions are stored in memory by examining the choice of data structures along with their advantages and disadvantages.

#### 5.3.3.1 Naive Data Storage

For many representations a simple dynamically resizable array is a reasonable starting point. For data with a pre-defined spatial partitioning two things are needed, the type of data to store and a conversion function from a spatial coordinate to an index coordinate. This is often used when building maps representing occupancy or signed distance values. For irregular data, only the type of data to store is needed, for example point clouds or surfels. The naive storage of ordered data using a mapping function and unordered pointcloud data is illustrated in Figure 5.9.

The benefit of this naive approach is that it is simple and provides fast random access. The trade-off is, that large amounts of memory can be required for such a representation. Also while read and modify operations are fast, changing the spatial dimension of the representation can be very costly as the content of the entire data structure needs to be copied.

### 5.3.3.2 Hash Map

A natural extension to address the limitations of the naive storage method described above is to use a hash table. This approach divides the map into shards, applies a *hash function* to convert the coordinates of each shard into a single value, and stores the sharded data in a table indexed by this hash value. The shards are typically chosen to represent map subregions with well-established coordinates, such as cubes in a regular grid. These cubes may correspond to individual voxels or fixed-size groups of voxels, referred to as voxel *blocks*. Alternatively, they can also store other elements like points, surfels, or mesh fragments contained in their respective subregions.

A hash table retains the fast  $\mathcal{O}(1)$  look-up time of a fixed array while allowing the map to grow dynamically without reallocation. Three key considerations must be addressed when using a hash table for dense map storage:

- 1 **Granularity of the sharding:** Smaller shards improve sparsity by allocating data only where necessary. However, the number of shards should not grow too large, as this reduces the hash table's insertion performance and memory efficiency. This trade-off is particularly relevant when hash maps are used to store properties that only exist along the surface.
- 2 **Hash function:** An ideal hash function distributes keys evenly across the table, even when the data is spatially adjacent, as is often the case in mapping scenarios.
- 3 **Collision resolution:** The method for handling hash collisions, whether through linear chaining (where each entry contains a linked list) or open addressing, significantly affects the performance of the hash table.

In most cases, hash tables offer a good balance of fast access and efficient insertion and deletion of data. However, they may require initial tuning to perform well for a given application.

### 5.3.3.3 Tree-based Data Structures

tree

Another option to efficiently store spatial data while only occupying memory for relevant parts of the environment is to use hierarchical, tree-based representations. Just like hash tables, trees generally enable efficient access and insertions. However, their unique strength is their hierarchical structure, which can be used to efficiently store multi-resolution data and speed up spatial operations such as nearest neighbor search. The most prominent tree variants are kD-trees [72], bounding volume hierarchies (BVH) [223], and octrees [749].

Among them, the octree efficiently searches neighbors with the capability to integrate novel measurements incrementally. The octree is a tree representing each node by a so-called *octant* that refers to a subspace. An octant is defined by a center  $c \in \mathbb{R}^3$  and an extent  $e \in \mathbb{R}$ , corresponding to an axis-aligned bounding-box. Each octant has potentially 8 child octants of extent  $\frac{1}{2}e$ , as depicted in Figure 5.10.

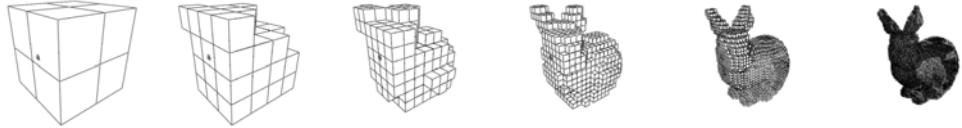


Figure 5.10 Example of an octree and its octants at different levels of the tree hierarchy. Each level of an octree subdivides the space in more fine-grained octants. Note that deeper levels of the octree only represent the occupied space.

Common practice is to store points only in the leaf octants (*i.e.*, octants without children) and determine subsets of points at inner octants of the tree structure by tree traversal.

To construct an octree, we iteratively divide space into octants within an axis-aligned bounding box encompassing a point cloud  $P$ . Each division splits  $P$  into subsets  $P_1, \dots, P_8$ , corresponding to 8 child octants of half extent  $\frac{1}{2}e$ . Non-empty subsets  $P_i$  form child octants with center  $c$  and extent  $\frac{1}{2}e$ , stopping at a specific octant size or a minimal point count. Once constructed, updates and insertions can efficiently be performed by traversing the tree structure and adding inner nodes as needed. When new data is inserted that falls outside of the tree's root octant, the tree can be extended by creating a new root node and assigning the new data and the old root node to its children.

In contrast to voxel grids, an octree represents only data-containing subspaces, enabling efficient storage of occupied space. However, this memory efficiency requires tree traversal to access specific leaf octants, potentially leading to increased runtime to locate points. Additionally, the tree structure itself must be explicitly represented, incurring extra memory overhead. Several recent approaches address these memory overheads [305, 817, 70].

#### 5.3.3.4 Hybrid Data Structures

To balance memory requirements and runtime for data access, several data structures combine the advantages of different data structures in specific ways, leading to hybrid representations. In this tradeoff, we accept less efficient memory usage but enable more efficient memory access.

For example, hashed voxel grids [805] combine the strengths of dense voxel grids and hash tables by splitting the environment into fixed-sized, dense blocks (*e.g.*  $8 \times 8 \times 8$  voxels), which are in turn stored in a hash table. Thanks to the hash table's flexibility, blocks only need to be allocated in locations that contain meaningful information (*e.g.* near the surface). At the same time, using a plain 3D array to store the voxels inside each block ensures that operations remain simple, efficient, and even suitable to GPU acceleration.

Another option is to combine hash tables with trees. In a similar vein to hashed

Section	Space Abstraction Type	Representing Map Entities
5.4.1	Points	Surface
5.4.2	Surfels	Surface
5.4.3	Mesh	Surface (connected)
5.4.4	Voxels	Occupancy or Implicit surface
5.4.5 - 5.4.6	Continuous function	Occupancy or Implicit surface

Table 5.1 *Summary of presented mapping methods.*

voxel grids, the VDB<sup>3</sup> data structure [789, 788] splits the space into hashed blocks, but stores a hierarchical tree inside each block. This data structure provides all the benefits of hierarchical trees, including multi-resolution representation and efficient nearest neighbor lookups. However, since each block has a fixed size, the maximum tree height is constant regardless of the size of the environment. Lookups and insertions can therefore be performed in constant time, and significantly faster than when using pure trees.

## 5.4 Constructing Maps: Methods and Practices

So far, we have explored the target quantities to estimate and the various space abstractions available for mapping. In this section, we will examine in detail the methods used to construct these map elements. The approaches are categorized by their main space abstraction, as shown in Table 5.1. Note that some of the methods use additional space abstractions to improve performance, for example, by grouping points into voxels for more efficient storage and faster queries.

### 5.4.1 Points

*points*

As mentioned in Section 5.3.2.1, naively storing points by accumulating the measured point clouds will not scale to large-scale environments and will lead to redundantly represented measurements. Therefore, most approaches [1266, 256, 1130] adopt a point-based representation in combination with a voxel grid or octree to represent the dense map. Moreover, the selection of a data structure is driven by the requirement for efficient nearest neighbor searches, essential for conducting scan registration through iterative closest point (ICP), where point correspondences must be iteratively established.

In order to handle large-scale environments, some methods, such as the well-known LiDAR SLAM LOAM [1266], filter the raw point clouds to extract corner and surface points thereby significantly reducing the point cloud size. A voxel grid

<sup>3</sup> VDB refers to sparse volumetric data and stands for several different thing as Voxel Data Base or Volumetric Data Blocks. Here we follow the terminology used in [789].

is applied to store only a subset of points in the map representation, pruning redundant measurements. Stemming from the point-based voxelization used in LOAM, several follow-up approaches [995, 1147, 836, 664, 891, 996] refine the extraction of points [995, 1147, 836], improve the optimization pipeline [836, 664], or integrate information of an IMU [891, 996].

Another branch of methods handles the amount of point cloud data differently to avoid reliance on a capable feature extraction approach. Regularly sampling the point clouds via a voxel grid [256] significantly reduces the number of points per LiDAR scan and removes potentially redundant information. The key insight is here that points in the voxel grid are not aggregated and averaged, but original measurements are retained. Following these insights, Dellenbach et al. [256] and Vizzo et al. [1130] use this strategy to downsample an input point cloud, only storing a restricted number of points inside a voxel grid map.

Overall, as also mentioned in Section 5.3.2.1, the (hashed) voxel grid serves dual purposes: it abstracts space by storing a limited number of point measurements per voxel, and it facilitates accelerated nearest neighbor search through direct indexing of neighboring voxels.

#### 5.4.2 Surfels

For surfels, similar strategies can be applied as for point clouds, but notably Stückler and Behnke [1038] and follow-up work by Droeuschel and Behnke [282] use an octree to represent surfels at multiple levels in the octree hierarchy for data association. The so-called multi-resolution surfel maps indirectly represent the surfels via accumulated mean and covariance statistics, like a NDT.

*surfels*

In contrast, Whelan et al. [1181] store surfels as a simple list and exploit efficient rendering techniques to produce a projection for data association for RGB-D SLAM in indoor environments. In this case, surfels are explicit geometric primitives and, therefore, need to be directly handled to update the surfel properties (*i.e.*, size and direction) accordingly [548]. A key contribution of Whelan et al. [1181] is leveraging a map deformation that directly deforms the surfels instead of relying on a PGO, which enables the use of the measurements represented by the surfels to deform the map on a loop closure detection. A similar strategy for map deformation of surfels was employed by Park et al. [843].

Similarly, Behley and Stachniss [66] target outdoor environments, which makes it necessary to represent the surfels via multiple submaps of  $100\text{ m} \times 100\text{ m}$  spatial extent that can be off-loaded from GPU memory. In contrast to ElasticFusion [1181], the approach relies on PGO but exploits that surfels can be freely positioned and ties surfels to poses enabling a straight-forward deformation of the map with pose-graph-optimized poses, which was also adopted by other approaches [1155].

### 5.4.3 Meshes

*meshes*

As introduced earlier, meshes offer an expressive, flexible way to represent connected surfaces. Mesh generation methods can be split into two families of approaches. The first family directly converts the measured points into a mesh. In contrast, the second family splits the problem into two steps: reconstructing an implicit surface, followed by iso-surface extraction to get the final mesh (see Section 5.3.2.6).

Methods in the first family typically work *directly* by computing the Delaunay triangulation of the input point set and identifying the subset of Delaunay triangles that lie on the surface. A detailed overview of such methods is provided in [157]. When building directly from points, the mesh implicitly adapts itself to the sampling density. This can be an advantage, as it provides adaptive resolution, but it also means these methods are more sensitive to sampling irregularities and holes. In practice, direct meshing methods are chosen when the entire surface can be sampled densely with a very accurate depth sensor, for example, using surveying equipment.

The second family of approaches uses an implicit surface as an *intermediate* step, to simplify the process of fusing and filtering the data before extracting the final surface mesh. One intuitive way to generate the implicit surface from data is to estimate the distance to the surface at each point on a regular grid. As described in Section 5.2.2, the implicit surface's sign must also be set according to whether each point is inside or outside an object. This information is often determined based on estimated surface normals, which can for example be obtained by applying Principal Component Analysis (PCA) over a small surrounding area. However, as indicated in [476], such methods may yield implicit surfaces that are discontinuous. Tackling this issue, Carr et al. [155] model the implicit surface using a collection of Radial Basis Functions (RBFs) and fit these to the input points by solving a global optimization problem. The resulting implicit surfaces are smooth by construction and faithfully fill holes based on the global context. Unfortunately, solving the underlying large, dense optimization problem is computationally expensive. Shen et al. [999] overcome this limitation by locally approximating the input points using moving least squares (MLS). Going one step further, Poisson Surface Reconstruction [544] fits the implicit function to the normals of the measured points by solving a partial differential equation (PDE), resulting in a sparse, computationally tractable optimization problem that is particularly robust to noise.

In robotics applications, constructing a mesh from a live sensor stream is often desirable. One way to make surface reconstruction efficient enough to run in real-time is to use incremental updates. TSDF-based surface reconstruction is particularly popular in practice given its inherently incremental nature and general simplicity. This method falls under the second family of approaches and estimates the implicit surface by averaging projective distances. Since the cost of updating the TSDF, or implicit surface in general, overshadows the cost of the mesh extraction, real-time methods primarily focus on optimizing the former.

### 5.4.4 Voxels

Voxel-based methods are among the most commonly used volumetric representations in 3D reconstruction and robotics. Instead of covering a swath of existing literature chronologically, this section will focus on concepts commonly encountered in practice and organize them according to three fundamental decision criteria: the chosen estimated quantity, data structure, and scalability considerations.

#### 5.4.4.1 Methods by their Estimated Quantity

The first choice in a voxel-based mapping framework is which quantity to estimate, with the most common options being *occupancy* (see Section 5.2.1) or a *distance* metric (see Section 5.2.2). The previous discussion in Section 5.2.3 can be used to decide between the two.

Since the introduction of the original continuous probabilistic occupancy measurement model for sonar [771], simplified piecewise-constant models have been developed to reduce computational costs [478]. This shift was influenced by the advent of LiDAR technology and the growing interest in transitioning from 2D to 3D maps. More recently, Loop et al. [689] presented a continuous probabilistic model that, instead of inflating objects, converges to an occupancy probability of 0.5 along objects' surfaces. Occupancy estimation, popular for collision avoidance due to its superior recall, is limited by its discontinuous nature and uninformative gradients compared to distance-based methods (see Section 5.2.3).

For distance metrics, we must not only estimate the positive part of the distance field but also extrapolate negative distances behind the surface since the surface is represented by the signed distance field's zero-crossings. To limit the accuracy impact of fusing imperfect positive and negative distances estimates (see Section 5.2.2), the updates are typically clamped to a small *truncation band* around the surface boundary. However, distance-based methods remain prone to erasing geometry. For example, when thin objects are observed from opposing sides, averaging the observed positive and hallucinated negative distances makes the zero-crossings flip around or disappear. Some works have analyzed the effect of the truncation band and weight drop-offs on the quality of the final reconstruction [134]. Fundamentally, the problem can be reduced but not eliminated. Overall, the surfaces estimated by TSDFs outperform occupancy methods along smooth surfaces at the cost of lower *recall* on thin objects.

The distance information provided by TSDFs is inherently valuable. However, instead of being conservative, TSDFs strictly overestimate the *Euclidean* distance. To address this safety concern, voxblox [819] popularized incrementally building ESDFs. Voxblox fuses the sensor data into a TSDF and then updates its ESDF using a brushfire algorithm [627]. Subsequently, FIESTA [426] proposed a hybrid approach that incrementally updates an ESDF map from an occupancy map instead.

#### 5.4.4.2 Methods by Data Structure

The simplest data structure for volumetric mapping is a static 3D array. As shown by KinectFusion [798], this data structure yields good results for small and fixed-size scenes. However, many applications require the ability to dynamically expand the map at runtime, while only allocating voxels where needed to save memory.

To address these concerns, Nießner et al. [805] proposed a voxel-block hashing scheme, which groups the voxels into blocks (*e.g.*,  $8 \times 8 \times 8$  voxels) that are stored in a hash-map. This data structure was quickly adopted for TSDFs, providing constant-time ( $\mathcal{O}(1)$ ) lookups and dynamic insertions. Of course, it can also be used to store occupancy probabilities, as shown by FIESTA [426]. Compared to hashing voxels individually, grouping them in blocks offers an adjustable trade-off between the hash table's size and the granularity at which voxels are allocated.

Naturally, voxels can also be stored using tree structures. Octomap [478] first popularized using an *octree* to store occupancy probabilities and has been the *de facto* standard for volumetric mapping for many years. A significant advantage of using trees is that they inherently support multi-resolution, while a major limitation is that encoding the tree's structure introduces a significant memory overhead, and that the cell lookup time is proportional to the tree's height. Most recent approaches address this limitation by leveraging hybrid data structures. Supereight [1122], for example, proposes to use a standard (dynamic) octree for the first levels and static octrees for the last few levels. These static octrees can be seen as octrees stored using a fixed-sized array. This removes the memory overhead of encoding parent-child relationships with pointers, at the cost of reducing granularity since static octrees are allocated as a block. The VDB [789] data structure was first introduced for the visual effects (VFX) industry and subsequently used by several volumetric mapping frameworks [718, 1129]. As discussed in Section 5.3.3.4, it combines block-hashing with trees to obtain the best of both worlds: good memory efficiency, hash-like constant time lookups and insertions, and tree-like multi-resolution.

A practical consideration is that downstream tasks often demand storing additional information, such as colors, semantics [398, 939] or an ESDF [819, 426] alongside the occupancy probabilities or TSDF. Although virtually any data structure can be extended to support additional channels, the required implementation effort scales with how complicated the underlying data structure is. This further motivates using simple data structures (*e.g.* voxel-block hashing) or flexible, third-party libraries.

#### 5.4.4.3 Methods by Measurement Integration Algorithm

The algorithm used to update the map based on depth measurements is referred to as the measurement integrator. It updates the estimated quantity for each observed voxel by applying the measurement model. The two main approaches used to integrate measurements are ray-tracing and projection-based methods.

For each measured point, ray tracing integrators cast a ray from the sensor to the point and update all the voxels intersected by the ray. An advantage of this approach is that it is very general, and only requires that the position of the sensor's origin is known. However, voxels may be hit by multiple rays, especially if they are near the sensor. This leads to duplicated efforts, and handling the resulting race conditions in parallel implementations creates implementation and performance overheads.

In contrast, projection-based methods directly iterate over the observed voxels and look up the ray(s) needed to compute their update by projecting each voxel into sensor coordinates. Iterating over the map instead of the rays inherently avoids race conditions. Projection-based methods are, therefore, prevalent in multi-threaded and GPU-accelerated volumetric mapping frameworks. The predictable access pattern resulting from directly iterating over the map also reduces memory bottlenecks. Yet, a major disadvantage is the need for explicit knowledge of the sensor's full pose and projection model. This method is also harder to use with disorganized point clouds, including the clouds obtained after applying LiDAR motion-undistortion.

#### *5.4.4.4 Methods by Scalability*

Memory and computational costs are two of the main bottlenecks in volumetric mapping. For fixed-resolution methods, the memory and computational complexities grow linearly with the map's total volume and cubically with the chosen resolution. Reducing these complexities is of significant research interest, as it is necessary to create detailed maps that scale beyond small, restricted volumes.

Early works in volumetric mapping mainly focused on reducing memory usage. For example, Octomap [478] proposes to use its octree's inner nodes to store their children's max or average occupancy. By recursively pruning out leaf nodes whose estimated quantities are close to their parent, constant areas in the map are automatically represented with fewer, lower resolution nodes. This adaptation to the environment's geometry is very effective in practice since environments predominantly consist of free space. Furthermore, storing min, max, or average values in the octree's inner nodes could be valuable for downstream tasks, as it enables map queries at lower resolutions and the use of hierarchical algorithms for tasks such as fast collision checking or exploration planning. Yet, a core limitation of Octomap is that it integrates all measurements at the highest resolution, meaning that the scaling of its computational complexity remains cubic.

Multi-resolution can also be leveraged to reduce the computational cost of measurement updates. Given that measurement rays are emitted at fixed angles, resulting in fewer rays hitting distant geometry, it seems logical to lower the update resolution as the distance increases. This can be achieved through multi-resolution ray-tracing [286] or multi-resolution projective integration [1122]. Supereight2 [347] reduces the computational complexity further by adjusting the update resolution to the entropy of the measurement updates. Such methods significantly enhance the update performance, yet a remaining challenge is that the map's different resolution

levels still have to be synchronized explicitly. One way to eliminate this synchronization requirement is to encode only the differences between each resolution level, instead of storing absolute values in each octree node. This can formally be done by applying wavelet decomposition. Wavelet-encoded maps can efficiently be queried at any resolution at any time. Using this property, wavemap [920] reduces the computational complexity even further by updating the map in a coarse-to-fine manner. In addition to adjusting the update resolution to the measurement entropy, it also skips uninformative updates, such as when the occupancy for an area in the map has converged to being free, and all measurements agree.

#### Gaussian Process

##### 5.4.5 Gaussian Processes

As mentioned in Section 5.3.2.5, formulating the mapping problem as a regression problem is desired to obtain a continuous representation. Moreover, if the aim is to limit the number of assumptions about the environment, solving a non-linear regression problem with non-parametric methods is ideal. A Gaussian process (GP) [906] is a stochastic, non-parametric, non-linear regression approach. It allows estimating the value of an unknown function at an arbitrary query point given noisy and sparse measurements at other points. We already learned in Chapter 2.2 how GPs can be used for continuous time trajectory representation. As will be apparent, GPs are also an appealing solution for mapping continuous quantities, and they have been extensively used in the robotics literature to model continuously spatial phenomena with depth sensors [1119, 813, 377, 573].

The information in GP models is contained in its mean  $\mathbf{m}(\mathbf{x})$  and kernel functions  $\mathcal{K}(\mathbf{x}, \mathbf{x}')$  and model the estimated continuous quantity as

$$f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{m}(\mathbf{x}), \mathcal{K}(\mathbf{x}, \mathbf{x}')). \quad (5.8)$$

Let  $\mathbb{X} = \{\mathbf{x}_j \in \mathbb{R}^D\}$  be a set of locations with measurements  $\mathbf{y}$ , with  $y_j = f(\mathbf{x}_j) + \epsilon_j$  of the estimated quantity taken at the locations  $\mathbf{x}_j$ . For  $J$  number of training pair  $(\mathbf{x}_j, y_j)$ , we assume the noise  $\epsilon_j$  to be *i.i.d* following Gaussian  $\epsilon_j \sim \mathcal{N}(\mathbf{0}, \sigma_j^2)$ . Given a set of testing locations  $\mathbb{X}^* = \{\mathbf{x}_n^* \in \mathbb{R}^D \mid n = 0, \dots, N\}$ , we can express the joint distribution of the function values and the observed target values as,

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\left(\mathbf{1}, \begin{bmatrix} \mathbf{K}(\mathbb{X}, \mathbb{X}) + \sigma_j^2 \mathbf{I} & \mathbf{K}(\mathbb{X}, \mathbb{X}^*) \\ \mathbf{K}(\mathbb{X}^*, \mathbb{X}) & \mathbf{K}(\mathbb{X}^*, \mathbb{X}^*) \end{bmatrix}\right), \quad (5.9)$$

where  $\mathbf{K} = [\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)]_{ij}$ . Thus the conditional distribution of  $(\mathbf{f}_* \mid \mathbb{X}, \mathbf{y}, \mathbb{X}^*) \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*))$ , with the mean equation is given by,

$$\bar{\mathbf{f}}_* = \mathbf{K}(\mathbb{X}^*, \mathbb{X}) [\mathbf{K}(\mathbb{X}, \mathbb{X}) + \sigma_j^2 \mathbf{I}]^{-1} \mathbf{y}, \quad (5.10)$$

and the covariance equation is,

$$\text{cov}(\mathbf{f}_*) = \mathbf{K}(\mathbf{X}^*, \mathbf{X}^*) - \mathbf{K}(\mathbf{X}^*, \mathbf{X}) [\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_j^2 \mathbf{I}]^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}^*). \quad (5.11)$$

Here, (5.10) and (5.11) are the predictive equations for the estimated quantitative.

GPs have proven particularly powerful to represent spatially correlated data, hence overcoming the traditional assumption of independence between cells, characteristic of the occupancy grid method for mapping environments. Gaussian Process Occupancy Map (GPOM)s [813] collects sensor observations and the corresponding labels (free or occupied) as training data; the map cells comprise testing locations, which are related to the training data as shown in (5.9). After the regression is performed using (5.10) and (5.11), the cell's probability of occupancy is obtained by “squashing” regression outputs into occupancy probabilities using binary classification functions.

In its original formulation GPOM is a batch mapping technique with cubic computational complexity ( $\mathcal{O}(J^3 + J^2 N)$ ). Approaches that aim to tackle this computational complexity—especially for incremental GP map building—have been proposed following this work, for example [573, 574, 1154, 378].

A key advantage of mapping with GP-based functions is that the estimated quantity can be linearly operated [966] and still produce a GP as an output. Given that derivatives and, therefore gradients, are linear operations, the differentiation output of the estimated quantity is probabilistic. A continuous representation of the uncertainty in the environment can be used to highlight unexplored regions and optimize a robot's search plan [378, 674]. The continuity property of the GP map can improve the flexibility of a planner by inferring directly on collected sensor data without being limited by the resolution of a grid/voxel cell.

#### 5.4.5.1 Gaussian Process Implicit Surface

Implicit surfaces can also be represented by a GP. Gaussian process implicit surface (GPIS) techniques [1183, 733, 674, 505] use a GP approach to estimate a probabilistic and continuous representation of the implicit surface given noisy measurements. Furthermore, GPIS can be also used to estimate not only the surface but also the distance field in a continuous manner [575, 1031, 634, 1192].

In the GPIS formulation, let us consider the distance field  $\mathbf{d}$  to be estimated from the distance to the nearest surface  $d_i$  given the points on the surface and its corresponding gradient  $\nabla \mathbf{d}$  computed through linear operators [966]. Then  $\mathbf{d}$  with  $\nabla \mathbf{d}$  can be modelled by the joint GP with zero mean (given that at the surface the distance is zero):

$$\begin{bmatrix} \mathbf{d} \\ \nabla \mathbf{d} \end{bmatrix} \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}')). \quad (5.12)$$

GPIS approaches have the ability to estimate a continuous implicit surface and the normal of the surface through the gradient, both with uncertainty. Some works

have considered the use of parametric function priors to capture given shapes more accurately [733, 505]. Other approaches aimed to estimate not only the implicit surface but the full distance field. Given the nature of the vanilla version of the GPIS formulation, the distance is well approximated near the measurements, *i.e.*, on the surface, but falls back to the mean, which in this case is zero, faraway from the surface. To estimate the full distance field in a continuous and probabilistic formulation further away from the surface, works have considered applying a non-linear operation to a GPIS-like formulation [1192, 1193, 632].

All these works have to deal with the computational complexity of the GP-based formulation, but as an exchange, a continuous, generative, and probabilistic representation of the environment, given only point clouds, can be achieved.

#### **5.4.6 Hilbert Maps**

##### *Hilbert maps*

Hilbert Maps (HMs) [901] are in many ways similar to GPOMs [813]. Both are continuous probabilistic models that do not discretize the space, unlike voxel-based methods, and in contrast to point-based methods are capable of interpolating missing data. As stated, the major challenge in GPOM is high computational expense. Thus the design goals of Hilbert maps were the following: (i) process data continuously in an online manner, (ii) model dependence between observations, and (iii) incorporate measurement uncertainty.

To achieve these goals, training a logistic regressor with stochastic gradient descent in a projected feature space is often leveraged. The classifier and optimizer combination enables online model updates using large amounts of data while the feature projection permits representing intricate spatial details with such a simple classifier.

The feature projection serves the same idea as the kernel in a GP, but instead of a full covariance we use an approximation. There are many options for this, including Nystroem [1182], Random Fourier Features [898], and Sparse Kernel [752], which is what we will be using. The goal of the sparse kernel is to limit the range at which observations have an influence which improves convergence and computational efficiency. The outcome is a kernel that drops to exactly 0 at a specific distance.

This kernel allows us to project points in 2D or 3D space into significantly higher dimensions by placing inducing kernels at regular intervals over the space to be mapped. Furthermore, this enables computing high-dimensional feature space representations of input data to be trained the logistic regression classifier using mini-batch stochastic gradient descent. Lastly, training is done by sampling free space points along the range measurement, while adding the return as an obstacle point.

One challenge faced by HMs is the expressivity of the used kernel. A radial basis function (RBF), as used in Figure 5.11, is a circle or a sphere and their values need to be combined to reconstruct intricate details of the environment. Therefore there

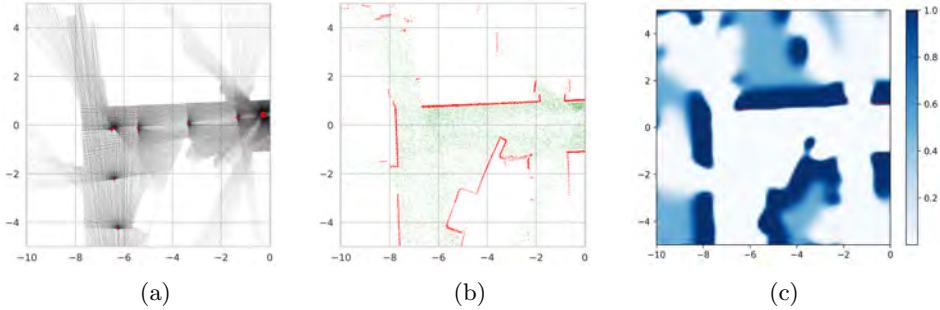


Figure 5.11 (a) The observations by a robot using a 2D LiDAR sensor, which is turned into a dataset of (b) Free (green) and occupied (red) points. These are then used to train a Hilbert Map as seen in (c).

is a trade-off in the form of number of kernels and their length-scale affecting the computational cost, reconstruction detail, and interpolation ability.

#### 5.4.7 Deep Learning in Mapping

With the recent interest in novel view synthesis using NeRFs (see Chapter 14 and [761]) —which provides compelling results for image generation via a simple multi-layer perceptron (MLP)— several approaches investigated the usage of neural representations to estimate a SDF. Learning to predict a SDF at arbitrary spatial locations leads to a continuous representation that can be turned into meshes at arbitrary resolutions, but can also lead to more complete representations due to the interpolation capabilities of the learned function. *NeRF*

Similar to implicit representations, this representation is learned from input data and approximated to provide a continuous function that can be queried at arbitrary locations. While often these neural representations are learned offline with given poses, there has been recent interest towards incremental approaches [1044, 1290] and approaches that estimate poses on-the-fly using a neural representation [1044, 965].

In particular, the approach of Sucar et al. [1044] uses a neural network to predict the SDF value of an arbitrary point in the scene based on RGB-D frames. Follow-up approaches extended this approach by separating the spatial representation of the features via voxel grids [859, 1306], octrees [1290], points [965, 261], etc. from the neural representation. In these approaches, small but descriptive features are stored in a spatial representation and used to determine with a small, neural network the SDF value of an arbitrary point in the scene. This allows decoupling the learned function from the spatial representation, which makes it possible to rely on small neural decoders to turn features into signed distance values, but also being effective for large-scale scenes, such as outdoor environments.

The area of deep learning-based mapping, reconstruction, and SLAM is rapidly evolving and integrates ideas from classical representations, such as surfel splatting [547, 737], to achieve remarkable results in terms of reconstruction quality and efficiency. Recent studies on feed-forward networks, such as Dust3r, for dense representations from monocular video will be illustrated in Chapter 13. In particular, the ability to render novel views and generate new data at arbitrary positions could be potentially exploited for robot learning without relying on simulated environments. For example, NeRF and Gaussian splatting [1315, 103] have gained considerable popularity, demonstrating significant potential in various SLAM-related works. These will be further detailed in Chapter 14.

## 5.5 Usage Considerations

All map representations trade off distinctive, often complementary, strengths and weaknesses. When choosing a map representation for a given application or robotic system, it is therefore important to carefully consider how the map will be used in all downstream tasks. Further factors to consider are the operating environment and available sensors. We will start by discussing environmental factors, which motivate several clear-cut choices, followed by more nuanced task-dependent considerations. Finally, we conclude this chapter with a brief discussion on usage considerations related to the existing methods presented in Section 5.4.

### 5.5.1 Environmental Aspects

Operating environments can be categorized as either *structured* or *unstructured*. In tightly controlled spaces, such as automated factories, custom map representations—tailored to the robot’s task and specific objects it will encounter—typically outperform general dense representations in terms of efficiency and accuracy. In contrast, the dense representations covered in this chapter can model objects of arbitrary shapes and work in any environment. When operating in changing or partially unknown environments, it is often important for robots to be able to distinguish observed free space from unobserved space. This information allows path planners to avoid unsafe motions through unobserved space, which could be occupied, and can also be used for exploration planning. Explicit surface representations, including points, surfels, and meshes, generally cannot distinguish between free and unobserved space, while all occupancy-based methods do. Implicit surface-based methods can also provide this distinction, though very often for more reconstruction-focused applications, this information is discarded farther from the surface to save computational and memory costs.

Another consideration is *scalability*. Explicit representations tend to be more memory efficient than implicit representations, as they only describe the surface itself and their fidelity can easily be adapted to the detail required for each part

of the scene. When free-space information is required, multi-resolution approaches can offer significant improvements over fixed-resolution voxelized representations in terms of accuracy, memory, and their ability to capture very thin objects.

One final consideration is whether the environment has a significant amount of *dynamic objects* and the degree to which these should be modeled. From the perspective of map representations, most existing approaches can be grouped into one of three categories. The first set of approaches does not consider dynamics and directly fuses all measurements into one of the representations introduced in this chapter. In practice, this might already suffice when using implicit representations, since their free-space updates typically do a good job at erasing leftovers of objects after they moved. The second category of approaches tries to only integrate the environment’s static elements into the map, by explicitly detecting and discarding all measurements corresponding to dynamic objects. This approach is particularly popular when using explicit maps, where leftovers are more tedious to remove, and generally makes it possible to generate clean maps even in highly dynamic spaces. The last set of solutions not only represents the background but also the moving elements in the scene. Note that this is commonly done using hybrid representations, mixing fundamental geometric representations introduced in this chapter with bespoke representations at the object level. For a detailed discussion on SLAM in dynamic environments, including concrete methods to implement the above and more advanced approaches, we refer the reader to Chapter 15.

### 5.5.2 Downstream Task Types

In addition to the environment, it is equally important to consider what map information is necessary for the robot’s required tasks. While any given operation can typically be performed on all representations, the efficiency and implementation complexity tend to vary greatly. The biggest difference lies in whether the operation is performed along the surface or in Cartesian space. As shown in Table 5.2, implicit representations generally allow for simple, efficient filtering of properties that are expressed in Cartesian coordinates, such as occupancy. In contrast, explicit representations are well suited to filter properties that are expressed along the surface, such as visual textures. This explains why explicit representations are generally more sensitive to the quality of the depth measurements, but can create very detailed, visually appealing 3D reconstructions. On the other hand, implicit methods are well suited for fusing noisy depth measurements, such as RGB-D camera data.

In terms of queries, explicit representations make it possible to directly iterate over the surface. This explains their popularity in rendering and graphics applications, and for tasks such as coverage path planning. However, they require additional steps, such as nearest neighbor lookups, to answer queries in Cartesian coor-

Operation	Efficient in	
	Explicit representation	Implicit representation
Filter measurements	Along the surface (texture,...)	In Cartesian space (occupancy,...)
Query and iterate	In surface coordinates (coverage planning,...)	In Cartesian coordinates (collision checking,...)
Modify surface	Geometry (deformation,...)	Topology (merge, cut, simplify,...)

Table 5.2 *Complementary strengths and weaknesses of explicit and implicit surface representations.*

dinates. The exact opposite is true for implicit representations, which are therefore commonly used for collision checking tasks.

Finally, explicit representations allow for efficient modifications of the surface’s geometry, including deformations. In practice, maps are often constructed by integrating depth measurements using pose estimates from an imperfect, drifting odometry system. Over time, the accumulated pose errors also lead to inconsistencies in the dense map. Just like in SLAM systems, these errors can be eliminated by deforming the dense map when detecting loop closures. Although both explicit and implicit surfaces can be deformed, this operation is inherently simpler and more efficient when using an explicit representation. In contrast, using an implicit representation simplifies and improves the efficiency of operations affecting the surface’s topology, or connectivity. Implicit representations are therefore often used to merge surface estimates, combine or subtract object shapes, and simplify surfaces.

It is important to remember that different representations can also be used in tandem to leverage their respective strengths. One good example of a hybrid approach is TSDF-based meshing (Section 5.4.3), where noisy depth measurements are first conveniently filtered using an implicit surface representation (TSDF) which is then converted to an explicit representation (mesh) using Marching Cubes. When deciding whether the advantages of hybrid representations outweigh the overhead they introduce, it is worth considering how the conversions can be limited to only happen locally and infrequently.

### 5.5.3 Summary of Mapping Methods

We now conclude our discussion by summarizing the key differences between the existing methods presented in this chapter. Starting with the explicit representations, using a collection of points to describe the surface is simple and requires the fewest assumptions, but it is also the least informative. Beyond infinitesimal points, surfels represent the surface’s properties over small neighborhoods, or patches. Finally, meshes explicitly represent the surface’s connectivity and allow its properties

to smoothly be interpolated. However, estimating the surface’s connectivity requires the most assumptions and sometimes comes at a significant computational cost.

In terms of implicit representations, a particular advantage of implicit surfaces over occupancy maps is that they offer fast, high-quality distance information and gradients which are beneficial for optimization-based planning. However, filtering occupancy estimates requires less assumptions and, for voxel-based methods, occupancy maps are better at capturing thin obstacles. In cases with particularly noisy or sparse depth measurements, non-voxelized implicit representations, based on GPs and Hilbert Maps, provide particularly good uncertainty estimates. As they explicitly consider the geometry’s spatial correlations, they are generally also better at interpolating partially observed surfaces.

One rapidly advancing research area is that of learning-based methods. In terms of learning-based implicit representations, NeRFs have been shown to enable promising new capabilities, particularly for semantic modeling and spatial reasoning. More recently, Gaussian splatting [551] —an explicit representation bearing similarities to surfels— led to an increasing interest into approaches using splatting [547, 737, 1258]. Researchers are actively working on improving the computational and memory footprint of these approaches, testing what new skills they can enable, and exploring how they can be integrated into complete robotic systems. Looking ahead, we expect learning-based methods to increase the generality and expressiveness of dense representations, while improving their ability to handle noisy measurements, incomplete observations, and dynamic objects through learned priors.

### Acknowledgments

The authors thank Lan Wu for her support in preparing this chapter.

# 6

## Certifiably Optimal Solvers and Theoretical Properties of SLAM

David M. Rosen, Kasra Khosoussi, Connor Holmes, Gamini Dissanayake,  
Timothy Barfoot, and Luca Carlone

Chapters 1–3 have discussed how to formalize the estimation problems arising in SLAM as optimization problems, using maximum a posteriori estimation or (more generally) *M-estimation*. Moreover, these chapters have introduced iterative local solvers (*e.g.*, Gauss-Newton, Levenberg-Marquardt, gradient descent) that look for a solution of the resulting optimization by iteratively refining a given initial guess.

In this chapter, we take a closer look at the optimization problems arising in SLAM and address two fundamental questions. First, can we design efficient algorithms that are guaranteed to compute a *globally* optimal solution of these problems, possibly without an initial guess? And second, *how accurate* is the optimal solution as an estimate of the ground truth, and what factors affect its accuracy?

The first question is about *computation* and *reliability*: the question starts from the observation that SLAM requires solving nonconvex optimization problems, which have multiple local minima. Depending upon the quality of the initial guess, iterative algorithms can get stuck in local minima, and hence produce incorrect estimates (Figure 6.1). Moreover, iterative methods do not provide tools to detect convergence to suboptimal solutions, which leads to trustworthiness and reliability concerns in practical applications. In Section 6.1, we show that despite the nonconvexity of typical SLAM problems, we can design *certifiably optimal algorithms* that solve the SLAM optimization to provable optimality, and can discern sub-optimal solutions from optimal ones. These algorithms are based on a mathematical tool, known as *semidefinite relaxation*, which we review in Section 6.1.1. The chapter presents certifiable algorithms for PGO (SE-Sync, Section 6.1.2), landmark-based SLAM with range and bearing measurements (Section 6.1.3), and then discusses extensions to other SLAM problems, including problems with range-only measurements, anisotropic noise, and outliers (Section 6.1.4).

The second question is concerned with *fundamental limits and estimation errors*: when designing a SLAM system, one often wants to ensure that the robot pose estimate and the map estimate are close to the ground truth, since large errors may induce failure in downstream tasks, including motion planning. Understanding fundamental limits is not only useful for analytical purposes but has very practical implications: as we discuss below, these limits are influenced by the structure of

the factor graph, and we can actively control this structure by carefully driving the robot (*e.g.*, forcing it to revisit places or landmarks). Therefore, in Section 6.2 we discuss information-theoretic limits on the achievable accuracy of the SLAM estimate. In particular, Section 6.2.1 introduces the Cramér–Rao Lower Bound to quantify the estimation error in SLAM, and Section 6.2.2 draws connections between this bound and the graphical structure of the SLAM problem.

As we will see, investigating these questions reveals deep connections between the algebraic, geometric, and graph-theoretic structures encoded in the SLAM problem, and the computational and statistical hardness of solving it. As usual, we conclude the chapter with an outlook to recent trends and references in Section 6.3.

## 6.1 Certifiably Optimal Solvers for SLAM

SLAM is conventionally formulated as a *high-dimensional* and *nonconvex* problem. Finding the global solution to a general nonconvex optimization problem is inherently challenging, primarily due to the existence of numerous local minima (Figure 6.1). In particular, many special cases of SLAM are known to be *NP-hard*, including, *e.g.*, angular synchronization, rotation averaging, and PGO. It follows that there is no algorithm that is capable of efficiently solving these problems in general, unless  $P = NP$  [935].

Despite this theoretical complexity, early SLAM research consistently demonstrated surprising convergence to solutions close to the ground truth. SLAM algorithms are typically initialized with robot poses computed using odometry. In a well-calibrated mobile robot, odometry drift can be reduced to less than one percent of the distance traveled. Thus, the unexpectedly reliable convergence of SLAM was initially attributed to the availability of high-quality initial estimates.

However, subsequent research has shown that even in the presence of poor or inconsistent initializations, a range of optimization techniques, including Stochastic Gradient Descent, Levenberg–Marquardt, and preconditioned conjugate gradient methods, can often recover near-optimal solutions. These findings suggest that, despite being a nonlinear and nonconvex problem, SLAM possesses an intrinsic structure that makes it particularly amenable to solution through specialized optimization strategies. At the same time, these results show that simply applying off-the-shelf *local* optimization methods (such as gradient descent or quasi-Newton methods) can produce egregiously wrong estimates, even if the underlying instance of the SLAM problem is well-posed. This insight has motivated the development of algorithmic frameworks that explicitly leverage the graph-theoretic and geometric underpinnings of the SLAM problem to obtain improved convergence properties and guarantees.

One of the most exciting recent advances in SLAM has been the development of practical optimization algorithms that—despite this problem’s general intractability—are nevertheless provably capable of recovering *certifiably globally optimal* solutions

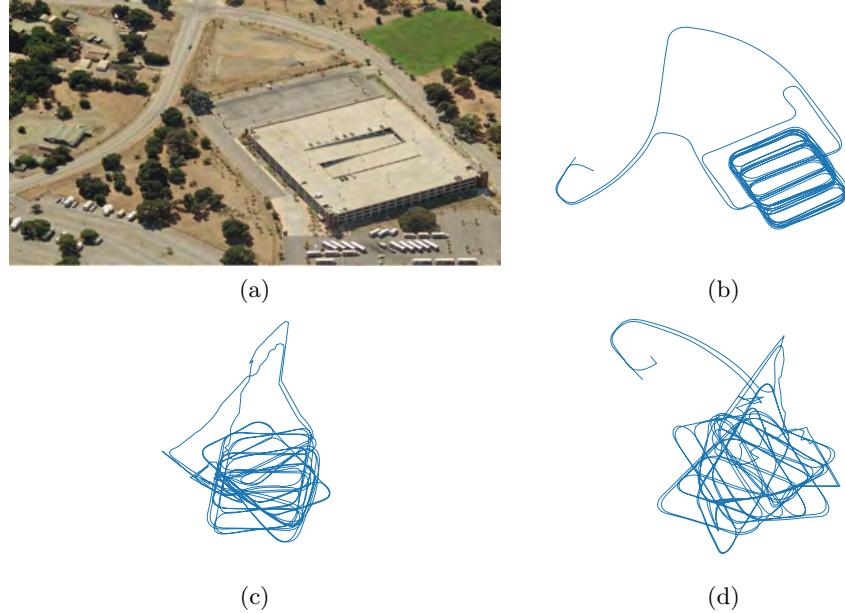


Figure 6.1 Examples of local minima in maximum likelihood estimation for SLAM. This figure shows several local minima for a real-world instance of the pose-graph SLAM problem (6.9) in which a robot navigated through the multi-level parking garage shown in (a). (b) shows the *correct* (i.e. *globally optimal*) solution of (6.9) computed using SE-Sync [934], while (c) and (d) show examples of suboptimal *local* minima obtained from a local optimization method initialized at a randomly-sampled starting point. (Panel (a) reproduced from [400]).

*convex relaxation*

of common SLAM formulations (*e.g.*, PGO) under mild conditions. These techniques, called *certifiably correct* optimization methods, are based on *convex relaxation* (rather than *local* optimization). Moreover, they are *certifiably optimal* in the sense that upon computing a solution to the optimization problem, they will be able to quantify how suboptimal that estimate is, and possibly certify its optimality. Such a statement does not contradict the NP-hardness of the problem: in worst-case scenarios, these algorithms can still fail to produce a certificate of optimality for a solution they compute and only provide a suboptimality bound. However, these algorithms remain of practical interest for two main reasons: (i) in practice, they *do* produce an optimality certificate for virtually all SLAM problems of practical interest (*e.g.*, for reasonable amounts of measurement noise), and (ii) the failure to produce an optimality certificate is in itself informative, since it might trigger a warning to downstream tasks not to trust the SLAM estimate, or for the robot to take fail-safe measures.

In this section we provide a brief introduction to certifiably correct estimation methods for SLAM. We begin by considering *Shor's relaxation*, one of the funda-

mental tools that is used to construct the convex relaxations underpinning certifiable estimation techniques. Next, we show how to apply Shor's relaxation to produce a certifiably correct estimation method (*SE-Sync*) to solve the fundamental problem of PGO. Finally, we discuss discuss extensions of SE-Sync and Shor's relaxation to a broader range of SLAM problems.

### 6.1.1 Shor's Relaxation

In this subsection we introduce *Shor's relaxation*, one of the fundamental tools that we will use to construct the convex relaxations underpinning certifiable estimators.

In brief, Shor's relaxation is a procedure for constructing convex relaxations of a *Quadratically constrained quadratic program (QCQP)*; that is, optimization problems in which the objective and constraint functions are *quadratics*. As we will see in the next section, many common SLAM formulations can be cast as QCQPs.

We introduce Shor's relaxation by describing its application to a generic QCQP, and then tailor it to SLAM in the next section. Consider the following QCQP:

$$\begin{aligned} p^* = \min_{\mathbf{x} \in \mathbb{R}^n} & \mathbf{x}^\top \mathbf{C} \mathbf{x} \\ \text{s.t. } & \mathbf{x}^\top \mathbf{A}_i \mathbf{x} = b_i \quad i = 1, \dots, m. \end{aligned} \tag{QCQP}$$

where  $\mathbf{C}, \mathbf{A}_1, \dots, \mathbf{A}_m \in \mathbb{S}^n$  are symmetric matrices and  $\mathbf{b} = (b_1, \dots, b_m) \in \mathbb{R}^m$  is a vector. We will show how to produce a convex relaxation of (QCQP) by applying a sequence of simple algebraic manipulations.

To begin, if  $\mathbf{M} \in \mathbb{S}^n$  is any symmetric matrix, we may exploit the cyclic property of the trace to rewrite the quadratic form  $\mathbf{x}^\top \mathbf{M} \mathbf{x}$  determined by  $\mathbf{M}$  according to

$$\mathbf{x}^\top \mathbf{M} \mathbf{x} = \text{tr}(\mathbf{x}^\top \mathbf{M} \mathbf{x}) = \text{tr}(\mathbf{M} \mathbf{x} \mathbf{x}^\top). \tag{6.1}$$

Applying (6.1) to (QCQP), we thus obtain the equivalent form:

$$\begin{aligned} p^* = \min_{\mathbf{x} \in \mathbb{R}^n} & \text{tr}(\mathbf{C} \mathbf{x} \mathbf{x}^\top) \\ \text{s.t. } & \text{tr}(\mathbf{A}_i \mathbf{x} \mathbf{x}^\top) = b_i, \quad i = 1, \dots, m. \end{aligned} \tag{6.2}$$

Now observe that the decision variable  $\mathbf{x}$  only enters problem (6.2) through outer products of the form  $\mathbf{X} \triangleq \mathbf{x} \mathbf{x}^\top$ ; note that every such matrix  $\mathbf{X}$  is *symmetric*, *rank-1*, and *positive-semidefinite* by construction. Conversely, if  $\mathbf{X} \in \mathbb{S}_+^n$  is a positive-semidefinite matrix and  $\text{rank}(\mathbf{X}) = 1$ , it is easily shown (by considering a symmetric eigendecomposition) that  $\mathbf{X}$  admits a symmetric factorization of the form  $\mathbf{X} = \mathbf{x} \mathbf{x}^\top$  for some  $\mathbf{x} \in \mathbb{R}^n$ . Putting these observations together, we thus have the equivalence

$$\mathbf{X} \in \mathbb{S}_+^n \text{ and } \text{rank}(\mathbf{X}) = 1 \iff \exists \mathbf{x} \in \mathbb{R}^n \text{ such that } \mathbf{X} = \mathbf{x} \mathbf{x}^\top. \tag{6.3}$$

In light of (6.3), problem (6.2) is equivalent to

$$\begin{aligned} p^* = \min_{\mathbf{X} \in \mathbb{S}^n} & \operatorname{tr}(\mathbf{C}\mathbf{X}) \\ \text{s.t. } & \operatorname{tr}(\mathbf{A}_i \mathbf{X}) = b_i, \quad i = 1, \dots, m, \\ & \mathbf{X} \succeq 0, \\ & \operatorname{rank}(\mathbf{X}) = 1. \end{aligned} \tag{6.4}$$

Thus far problems (6.2) and (6.4) are completely equivalent; however, formulation (6.4) has the advantage that it reveals a great deal of useful structure. Indeed, the objective and constraint functions in (6.4) are *linear* functions of the (matrix) decision variable  $\mathbf{X}$ , and the positive-semidefiniteness constraint  $\mathbf{X} \succeq 0$  is *convex*. Thus, the only difficulty in solving (6.4) is due to the (nonconvex) rank constraint.

*Shor's relaxation* [1008] simply consists of discarding the rank constraint appearing in (6.4), thereby producing the following *convex relaxation* of (QCQP):

$$\begin{aligned} d^* = \min_{\mathbf{X} \in \mathbb{S}^n} & \operatorname{tr}(\mathbf{C}\mathbf{X}) \\ \text{s.t. } & \operatorname{tr}(\mathbf{A}_i \mathbf{X}) = b_i, \quad i = 1, \dots, m, \\ & \mathbf{X} \succeq 0. \end{aligned} \tag{SDP}$$

*semidefinite program*

Note that (SDP) entails minimizing a linear function over the set of positive-semidefinite matrices, subject to a set of linear equality constraints; problems of this form are called *semidefinite programs*. Semidefinite programs, or SDPs, are convex optimization problems, and can be solved in polynomial time.

Now let us consider the relation between problem (QCQP) and its convex relaxation (SDP). First, note that we obtained (QCQP) from (SDP) by *expanding the former's feasible set* (*i.e.*, by dropping the rank constraint in (6.4)); indeed, it is easy to see that every feasible point  $\mathbf{x} \in \mathbb{R}^n$  in (QCQP) *lifts* to a corresponding feasible point  $\mathbf{X} \triangleq \mathbf{x}\mathbf{x}^\top$  for (SDP).<sup>1</sup> It follows that the optimal values of (QCQP) and (SDP) satisfy the relation:

$$d^* \leq p^*, \tag{6.5}$$

since the latter problem minimizes the same objective over a larger feasible set.

Inequality (6.5) already provides a very useful method for *assessing the quality* of candidate solutions of (QCQP). Suppose that we have a feasible point  $\hat{\mathbf{x}} \in \mathbb{R}^n$  of (QCQP); for example,  $\hat{\mathbf{x}}$  might have been obtained by performing *local* optimization. Writing  $f(\mathbf{x}) \triangleq \mathbf{x}^\top \mathbf{C} \mathbf{x}$  for the objective, inequality (6.5) implies that we may bound the *suboptimality*  $f(\hat{\mathbf{x}}) - p^*$  of  $\hat{\mathbf{x}}$  as a solution of (QCQP) according to

$$f(\hat{\mathbf{x}}) - p^* \leq f(\hat{\mathbf{x}}) - d^*. \tag{6.6}$$

<sup>1</sup> Dropping the rank constraint causes the feasible set to go from dimension  $n$  to dimension  $\frac{n(n+1)}{2}$ . This makes the term *lift* more concrete and motivates both why the convex relaxation can circumvent the non-convexity of the original problem, while possibly requiring specialized solvers to cope with the increased dimensionality.

Note that while the optimal value of  $p^*$  of (QCQP) is very hard to compute in general, the optimal value  $d^*$  can be computed efficiently by solving the relaxation (SDP). Inequality (6.6) thus gives us a practical way of *bounding*  $\hat{\mathbf{x}}$ 's suboptimality without the need to know  $p^*$  itself. In particular, if the right-hand side of (6.6) is small, we may conclude that  $\mathbf{x}$  is a *near-optimal* solution of (QCQP).

Moreover, if we solve the relaxation (SDP) and it so happens that the resulting minimizer  $\mathbf{X}^* = \mathbf{x}^* \mathbf{x}^{*\top}$  has rank 1, then it immediately follows that the vector  $\mathbf{x}^* \in \mathbb{R}^n$  is a *global minimizer* for the original (*nonconvex*) problem (QCQP) (since  $\mathbf{x}^*$  is feasible in (QCQP) and satisfies  $f(\mathbf{x}^*) = d^*$  in (6.6)). As we will see, it turns out that this favorable situation actually occurs quite often for many robotic state estimation tasks, enabling us to recover *exact, globally optimal* solutions for the *nonconvex* problem (QCQP) from solutions of its relaxation (SDP).

In practice, solving large SDPs might still be slow or memory intensive with off-the-shelf solvers, and it is common to develop specialized solvers (*e.g.*, SE-Sync, which we describe below) for large-scale robotics applications.

### 6.1.2 SE-Sync: Certifiably Correct Pose-Graph Optimization

In this subsection we show how one can apply Shor's relaxation to develop a certifiably correct algorithm (*SE-Sync*) for solving the fundamental problem of pose-graph optimization (PGO). PGO is one of the simplest and most commonly used SLAM formulations, and thus provides a natural concrete example to illustrate the construction of certifiable estimation algorithms. Moreover, PGO was the first SLAM formulation that was shown to be amenable for convex relaxations [149, 151, 936], and the corresponding ideas have been shown to generalize to a wide range of SLAM problems, as we will discuss below.

Our development proceeds in three stages. First, we show how to formalize PGO as an instance of maximum likelihood estimation, and how to reduce it to a QCQP. Next, we derive Shor's SDP relaxation for PGO, and (crucially) show that this relaxation is in fact *exact* for sufficiently small measurement noise; this implies that we can recover *globally optimal* solutions to PGO by solving its (convex) SDP relaxation. Finally, we describe a specialized, structure-exploiting optimization algorithm that enables us to solve large-scale instances of this SDP relaxation in practice.

#### 6.1.2.1 Pose-Graph Optimization: QCQP Formulation

Pose-graph optimization (PGO) estimates the values of a set of  $n$  unknown poses  $\mathbf{T}_1, \dots, \mathbf{T}_n \in \text{SE}(d)$  in  $d$ -dimensional space (typically in SLAM  $d = 2$  or  $3$ ), given noisy measurements  $\tilde{\mathbf{T}}_{ij} \approx \mathbf{T}_i^{-1} \mathbf{T}_j$  of a set of relative pose measurements between them. In practice, the unknown poses  $\mathbf{T}_1, \dots, \mathbf{T}_n$  describe the trajectory of the robot (*i.e.*, they are sampled at discrete times along the robot trajectory), while the measurements  $\tilde{\mathbf{T}}_{ij}$  are obtained by the SLAM front-end, *e.g.*, through LiDAR scan

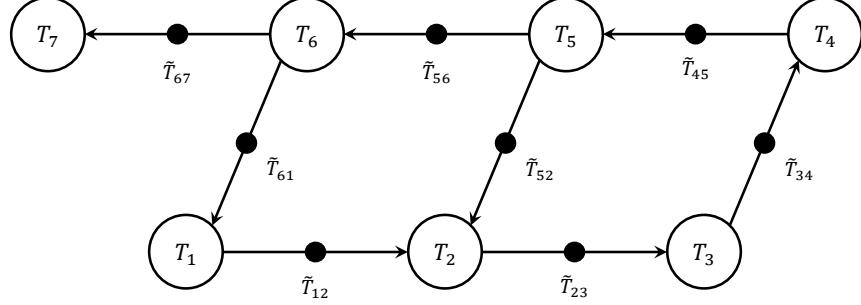


Figure 6.2 An example of pose graph. Here the vertices are in one-to-one correspondence with the unknown poses  $\mathbf{T}_i = (\mathbf{t}_i, \mathbf{R}_i) \in \text{SE}(d)$  to be estimated, and the directed edges are in one-to-one correspondence with the set of noisy measurements  $\tilde{\mathbf{T}}_{ij} \approx \mathbf{T}_i^{-1}\mathbf{T}_j$  of the relative poses between them.

matching, wheel odometry, or 3D computer vision techniques. In this subsection, we show how to formalize this estimation problem via maximum likelihood estimation. As we will see, under suitable assumptions on the noise, the resulting optimization problem is a QCQP.

To begin, it is often convenient to model the data defining this estimation problem using a *pose graph*  $\vec{\mathcal{G}}$ ,<sup>2</sup> constructed as follows. Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a simple undirected graph whose nodes  $i \in \mathcal{V}$  are in one-to-one correspondence with the unknown poses  $\mathbf{T}_i$  and whose edges  $\{i, j\} \in \mathcal{E}$  are in one-to-one correspondence with the set of available measurements.<sup>3</sup> We will assume (without loss of generality) that  $\mathcal{G}$  is connected.<sup>4</sup> The pose graph  $\vec{\mathcal{G}} = (\mathcal{V}, \vec{\mathcal{E}})$  is then obtained from  $\mathcal{G}$  by assigning an orientation for each of the latter's edges (cf. Figure 6.2). By convention, the measurement  $\tilde{\mathbf{T}}_{ij}$ , which describes (a noisy version of) the pose  $\mathbf{T}_j$  in the coordinate frame of the pose  $\mathbf{T}_i$ , is associated with a directed edge from  $i$  to  $j$ .

In order to formalize PGO as a maximum likelihood estimation, we must posit a noise model for the available measurements  $\{\tilde{\mathbf{T}}_{ij}\}$ . To do so, we will make use of the *isotropic Langevin distribution*  $\mathcal{L}(\mathbf{M}, \kappa)$ : this is an exponential family distribution over  $\text{SO}(d)$  whose probability density function is given by

$$p(\mathbf{R}; \mathbf{M}, \kappa) = \frac{1}{c_d(\kappa)} \exp(\kappa \text{tr}(\mathbf{M}^\top \mathbf{R})), \quad (6.7)$$

where  $\mathbf{M} \in \text{SO}(d)$  and  $\kappa \geq 0$  are parameters, and  $c_d(\kappa)$  is a normalization constant. Note that  $\mathbf{M}$  plays the role of a *location* parameter (called the *mode*), while  $\kappa \geq 0$

<sup>2</sup> A pose graph is a special instance of a factor graph, where the variable nodes are poses and the factor nodes relate pairs of poses.

<sup>3</sup> The assumption that  $\mathcal{G}$  is simple is not actually required for our methods, but will help to ease notational burden by avoiding the need to distinguish between multiple parallel edges.

<sup>4</sup> If  $\mathcal{G}$  is not connected, then the PGO problem decomposes into a set of *independent* estimation problems over the connected components of  $\mathcal{G}$ ; thus, the general case is always reducible to the case of a connected graph.

is a scalar *concentration* parameter. The isotropic Langevin distribution admits a particularly simple generative description in dimensions 2 and 3: to produce a sample  $\tilde{\mathbf{R}} \sim \mathcal{L}(\mathbf{M}, \kappa)$ , we first sample a rotation *angle*  $\theta \sim \text{vonMises}(0, 2\kappa)$  from the von Mises distribution on the circle, and then set  $\tilde{\mathbf{R}} = \mathbf{M} \cdot \mathbf{R}(\theta)$  if  $d = 2$ , or  $\tilde{\mathbf{R}} = \mathbf{M} \exp(\theta \mathbf{v}^\wedge)$  if  $d = 3$ , where  $\mathbf{v} \sim \mathcal{U}(S^2)$  is a uniformly sampled rotation *axis* [934]. Intuitively, one can think of this distribution as an analogue of the Gaussian distribution over the (non-Euclidean) manifold of rotations.

Given a pose-graph  $\vec{\mathcal{G}} = (\mathcal{V}, \vec{\mathcal{E}})$ , we will assume that each measurement  $\tilde{\mathbf{T}}_{ij} = (\tilde{\mathbf{t}}_{ij}, \tilde{\mathbf{R}}_{ij}) \in \text{SE}(d)$  is obtained by sampling from the following probabilistic generative model:

$$\begin{aligned}\tilde{\mathbf{t}}_{ij} &= \bar{\mathbf{t}}_{ij} + \mathbf{t}_{ij}^\epsilon, & \mathbf{t}_{ij}^\epsilon &\sim \mathcal{N}(0, \tau_{ij}^{-1} I_d), & \forall (i, j) \in \vec{\mathcal{E}}, \\ \tilde{\mathbf{R}}_{ij} &= \bar{\mathbf{R}}_{ij} \mathbf{R}_{ij}^\epsilon, & \mathbf{R}_{ij}^\epsilon &\sim \mathcal{L}(\mathbf{I}_d, \kappa_{ij}),\end{aligned}\quad (6.8)$$

where  $\bar{\mathbf{T}}_{ij} = (\bar{\mathbf{t}}_{ij}, \bar{\mathbf{R}}_{ij}) \in \text{SE}(d)$  is the true (latent) value of the relative pose between  $\mathbf{T}_i$  and  $\mathbf{T}_j$ . Model (6.8) assumes the translation component  $\tilde{\mathbf{t}}_{ij}$  of the  $ij$ th measurement is corrupted by *additive mean-zero isotropic Gaussian noise* with concentration parameter  $\tau_{ij} > 0$ , and the rotational component  $\tilde{\mathbf{R}}_{ij}$  is corrupted by *multiplicative isotropic Langevin noise* with mode  $\mathbf{I}_d$  and concentration parameter  $\kappa_{ij} \geq 0$ .

The primary motivation behind our use of the noise model (6.8) (as opposed to the more ‘generic’ exponentiated-Gaussian noise model over general Lie groups mentioned in Chapter 2) is that its associated maximum likelihood estimation takes a particularly simple algebraic form. Indeed, given a set of noisy measurements  $\tilde{\mathbf{T}}_{ij}$  sampled from (6.8), a straightforward calculation shows that the associated maximum likelihood estimation is

*Problem 1* (Pose-Graph Optimization)

$$p_{\text{MLE}}^* = \min_{\substack{\mathbf{t}_i \in \mathbb{R}^d \\ \mathbf{R}_i \in \text{SO}(d)}} \sum_{(i, j) \in \vec{\mathcal{E}}} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2. \quad (6.9)$$

In particular, note that the objective appearing in (6.9) is a simple (quadratic) linear least-squares loss. Moreover, the constraints  $\mathbf{R}_i \in \text{SO}(d)$  can be written as quadratic constraints in the planar ( $d = 2$ ) and three-dimensional case ( $d = 3$ ). In the next subsection, we will exploit this fact to derive a convex relaxation of Problem 1 via Shor’s relaxation.

#### 6.1.2.2 Applying Shor’s Relaxation to Pose-Graph Optimization

In this subsection we show how one can apply Shor’s relaxation to derive a convex relaxation of the PGO Problem 1.

**Simplifying the Maximum Likelihood Estimator.** Our first step will be to rewrite Problem 1 in a more compact form that only involves rotations, and reveals the correspondence between the optimization problem (6.9) and the underlying

graphs  $\mathcal{G}$  and  $\overrightarrow{\mathcal{G}}$  from which it is constructed. To that end, we introduce several matrices constructed from these graphs, and refer the reader to the box on the next page for a primer on (algebraic) graph theory (specifically, the *incidence* and *Laplacian* matrices).

**Elements of Algebraic Graph Theory.** Algebraic graph theory studies how to use algebra (e.g., matrices, vectors) to represent, analyze, and extract information from graphs. Here we review some basic concepts to support the explanations and connections drawn in this chapter. We also point the reader to Chapter 1, which draws connections between graphs and probabilistic graphical models.

A *directed graph*  $\vec{\mathcal{G}}$  is a pair  $(\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  is a finite set of *nodes*, and  $\mathcal{E}$  is a set of *edges*, where each edge contains an ordered pair of nodes. An edge  $e \in \mathcal{E}$  is in the form  $e = (i, j)$ , meaning that edge  $e$ , incident on nodes  $i$  and  $j$ , leaves node  $i$  and is directed towards node  $j$  ( $i$  is the *tail* of the edge, and  $j$  is the *head*).

For a graph with  $n$  nodes and  $m$  edges, the incidence matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  of a directed graph  $\vec{\mathcal{G}}$  is a matrix with entries in  $\{-1, 0, +1\}$  that describes the structure of the graph. Each row of  $\mathbf{A}$  corresponds to an edge, and the column corresponding to edge  $e = (i, j)$  has only two non-zero elements, one on the  $i$ -th column (equal to  $-1$ ) and the other on the  $j$ -th column (equal to  $+1$ ). While a more common definition of the incidence matrix has the edges on the columns (i.e., the transpose of our  $\mathbf{A}$ ), here we use this definition to keep some symmetry with the Jacobian matrix in Chapter 1. For instance, the incidence matrix of the graph in Figure 6.2 is:

$$\mathbf{A} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ -1 & +1 & & & & & \\ & -1 & +1 & & & & \\ & & -1 & +1 & & & \\ & & & -1 & +1 & & \\ & & & & -1 & +1 & \\ & & +1 & & & -1 & +1 \\ +1 & & & & & -1 & \end{bmatrix} \begin{array}{l} e_{12} \\ e_{23} \\ e_{34} \\ e_{45} \\ e_{56} \\ e_{67} \\ e_{52} \\ e_{61} \end{array} \quad (6.10)$$

The *Laplacian* matrix  $\mathbf{L} \in \mathbb{R}^{n \times n}$  is defined as  $\mathbf{L} \triangleq \mathbf{A}^\top \mathbf{A}$  and also captures the connectivity of the graph. In particular, the  $i$ -th diagonal element of  $\mathbf{L}$  corresponds to the node degree of the  $i$ -th node in the graph (i.e., the number of nodes connected to node  $i$ ), while an off-diagonal element in position  $(i, j)$  is equal to  $-1$  if there is an edge (regardless of its orientation) connecting node  $i$  and  $j$  or is zero otherwise. For instance, the Laplacian matrix of the graph in Figure 6.2 is:

$$\mathbf{L} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ +2 & -1 & & & & -1 & \\ -1 & +3 & -1 & & -1 & & \\ & -1 & +2 & -1 & & & \\ & & -1 & +2 & -1 & & \\ & & & -1 & +3 & -1 & \\ -1 & & & & -1 & +3 & -1 \\ & & & & & -1 & +1 \end{bmatrix} \begin{array}{l} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array} \quad (6.11)$$

We remark that the Laplacian matrix no longer captures the directionality of the edges in the graph since the off-diagonal entries are  $-1$  when there is an edge connecting the corresponding nodes regardless of its orientation.

The smallest eigenvalue of the Laplacian matrix is always equal to zero and the corresponding eigenvector is the vector with all entries equal to one (this can be easily seen from the fact that the entries on each row sum up to zero, hence  $\mathbf{L} \cdot \mathbf{1} = \mathbf{0}$ ). It turns out that the number of zero eigenvalues of the Laplacian corresponds to the number of connected components in the graph: a connected graph (where there is path of edges between any pair of nodes, regardless of the orientation of the edges) has a single zero eigenvalue, a graph formed by two disconnected subgraphs has two zero eigenvalues, etc. Moreover, for a connected graph, the *second* smallest eigenvalue is a measure of how well-connected the graph is, and is also known as the *algebraic connectivity* or *Fiedler value* of the graph.

Let us define some key matrices that are related to the graph Laplacian matrix and that will be used in our derivation. We define the *translational weight graph*  $\mathcal{W}^\tau = (\mathcal{V}, \mathcal{E}, \{\tau_{ij}\})$  to be the weighted undirected graph with node set  $\mathcal{V}$ , edge set  $\mathcal{E}$ , and edge weights  $\tau_{ij}$  for  $\{i, j\} \in \mathcal{E}$ , and let  $\mathbf{L}(\mathcal{W}^\tau) \in \mathbb{S}_+^n$  denote its Laplacian:

$$\mathbf{L}(\mathcal{W}^\tau)_{ij} = \begin{cases} \sum_{\{i,k\} \in \mathcal{E}} \tau_{ik}, & i = j, \\ -\tau_{ij}, & \{i, j\} \in \mathcal{E}, \\ 0, & \{i, j\} \notin \mathcal{E}. \end{cases} \quad (6.12)$$

This is simply a weighted version of the Laplacian matrix we defined above. Similarly, we write  $\mathbf{L}(\tilde{G}^\rho) \in \mathbb{S}_+^{dn}$  for the *connection Laplacian* determined by the rotational measurements  $\tilde{\mathbf{R}}_{ij}$  and precisions  $\kappa_{ij}$ ; this is the symmetric  $(d \times d)$ -block-structured matrix defined by

$$\mathbf{L}(\tilde{G}^\rho)_{ij} \triangleq \begin{cases} \left( \sum_{\{i,k\} \in \mathcal{E}} \kappa_{ik} \right) \mathbf{I}_d, & i = j, \\ -\kappa_{ij} \tilde{\mathbf{R}}_{ij}, & (i, j) \in \vec{\mathcal{E}}, \\ -\kappa_{ji} \tilde{\mathbf{R}}_{ji}^\top, & (j, i) \in \vec{\mathcal{E}}, \\ \mathbf{0}_{d \times d}, & \{i, j\} \notin \mathcal{E}. \end{cases} \quad (6.13)$$

We also define a few matrices constructed from the set of translation observations  $\tilde{\mathbf{t}}_{ij}$ . We let  $\tilde{\mathbf{V}} \in \mathbb{R}^{n \times dn}$  be the  $(1 \times d)$ -block-structured matrix with  $(i, j)$ -blocks determined by

$$\tilde{\mathbf{V}}_{ij} \triangleq \begin{cases} \sum_{\{k \in \mathcal{V} | (j, k) \in \vec{\mathcal{E}}\}} \tau_{jk} \tilde{\mathbf{t}}_{jk}^\top, & i = j, \\ -\tau_{ji} \tilde{\mathbf{t}}_{ji}^\top, & (j, i) \in \vec{\mathcal{E}}, \\ \mathbf{0}_{1 \times d}, & \text{otherwise,} \end{cases} \quad (6.14)$$

Let  $\tilde{\mathbf{D}} \in \mathbb{R}^{m \times dn}$  be the  $(1 \times d)$ -block-structured matrix with rows and columns indexed by  $e \in \vec{\mathcal{E}}$  and  $k \in \mathcal{V}$ , respectively, and whose  $(e, k)$ -block is given by

$$\tilde{\mathbf{D}}_{ek} \triangleq \begin{cases} -\tilde{\mathbf{t}}_{kj}^\top, & e = (k, j) \in \vec{\mathcal{E}}, \\ \mathbf{0}_{1 \times d}, & \text{otherwise,} \end{cases} \quad (6.15)$$

and  $\Omega \triangleq \text{Diag}(\tau_{e_1}, \dots, \tau_{e_m}) \in \mathbb{S}^m$  denote the diagonal matrix constructed from the translation measurement precisions. Finally, we also aggregate the rotation and translation state estimates into the block matrices  $\mathbf{R} \triangleq (\mathbf{R}_1 \ \dots \ \mathbf{R}_n) \in \text{SO}(d)^n \subset \mathbb{R}^{d \times dn}$  and  $\mathbf{t} \triangleq (\mathbf{t}_1 \ \dots \ \mathbf{t}_n) \in \mathbb{R}^{dn}$ .

With these definitions in hand, let us return to Problem 1. Observe that if we fix a value of the rotational states  $\mathbf{R}_1, \dots, \mathbf{R}_n$ , problem (6.9) reduces to a *linear least-squares* problem in the remaining translational decision variables  $\mathbf{t}_1, \dots, \mathbf{t}_n \in \mathbb{R}^d$ . Consequently, we may solve for an optimal assignment  $\mathbf{t}^*(\mathbf{R})$  of the translational states as functions of the rotational states  $\mathbf{R}$ :

$$\mathbf{t}^*(\mathbf{R}) = -\text{vec} \left( \mathbf{R}^* \tilde{\mathbf{V}}^\top \mathbf{L}(\mathcal{W}^\tau)^\dagger \right). \quad (6.16)$$

By substituting the optimal assignment (6.16) into (6.9), we may thus *analytically eliminate* the translational states from the pose-graph SLAM MLE, producing the following *simplified* (but equivalent) problem involving only the rotational states:

*Problem 2* (Rotation-Only Pose-Graph Optimization)

$$p_{\text{MLE}}^* = \min_{\mathbf{R} \in \text{SO}(d)^n} \text{tr}(\tilde{\mathbf{Q}} \mathbf{R}^\top \mathbf{R}) \quad (6.17a)$$

$$\tilde{\mathbf{Q}} = \mathbf{L}(\tilde{\mathbf{G}}^\rho) + \tilde{\mathbf{D}}^\top \boldsymbol{\Omega}^{\frac{1}{2}} \boldsymbol{\Pi} \boldsymbol{\Omega}^{\frac{1}{2}} \tilde{\mathbf{D}}, \quad (6.17b)$$

where  $\boldsymbol{\Pi} \in \mathbb{R}^{m \times m}$  is the matrix of the orthogonal projection  $\pi: \mathbb{R}^m \rightarrow \ker(\mathbf{A}(\vec{\mathcal{G}})\boldsymbol{\Omega}^{\frac{1}{2}})$  onto the kernel of the *weighted* incidence matrix  $\mathbf{A}(\vec{\mathcal{G}})\boldsymbol{\Omega}^{\frac{1}{2}}$  of  $\vec{\mathcal{G}}$ .

We observe that (6.17) involves only  $n$  *rotation matrices* (rather than  $n$  *poses*) and the problem now resembles the standard problem of multiple rotation averaging [435] (but with a more involved expression for the data matrix  $\tilde{\mathbf{Q}}$ ). On the more technical side, we note that although  $\boldsymbol{\Pi}$  is generically dense, by exploiting the fact that it is derived from the graph  $\vec{\mathcal{G}}$ , one can show that it admits the decomposition:

$$\boldsymbol{\Pi} = \mathbf{I}_m - \boldsymbol{\Omega}^{\frac{1}{2}} \bar{\mathbf{A}}(\vec{\mathcal{G}})^\top \mathbf{L}^{-\top} \mathbf{L}^{-1} \bar{\mathbf{A}}(\vec{\mathcal{G}}) \boldsymbol{\Omega}^{\frac{1}{2}} \quad (6.18)$$

where  $\bar{\mathbf{A}}(\vec{\mathcal{G}})\boldsymbol{\Omega}^{\frac{1}{2}} = \mathbf{L}\mathbf{Q}_1$  is a thin LQ decomposition of  $\bar{\mathbf{A}}(\vec{\mathcal{G}})\boldsymbol{\Omega}^{\frac{1}{2}}$  and  $\bar{\mathbf{A}}(\vec{\mathcal{G}})$  is the *reduced incidence matrix* of  $\vec{\mathcal{G}}$  obtained by deleting one of  $\mathbf{A}(\vec{\mathcal{G}})$ 's rows. Note that expression (6.18) requires only the lower-triangular factor  $\mathbf{L}$ , which will be sparse whenever the underlying graph  $\vec{\mathcal{G}}$  is, and can be obtained efficiently in practice. The sparse decomposition (6.17b)–(6.18) of the data matrix  $\tilde{\mathbf{Q}}$  will play a critical role in our implementation of efficient optimization methods (*cf.* Section 6.1.2.3).

**Forming the Relaxation.** Now we derive the semidefinite relaxation of Problem 1 that we will solve in practice, taking advantage of the simplified form (6.17).

We begin by relaxing the condition  $\mathbf{R} \in \text{SO}(d)^n$  to  $\mathbf{R} \in \text{O}(d)^n$ . The advantage of the latter condition versus the former is that since orthogonal matrices are defined by a set of (quadratic) orthonormality constraints, the orthogonal relaxation of Problem 2 is a homogeneous QCQP. Indeed, writing

$$\begin{aligned} \text{BlockDiag}_{d \times d}: \mathbb{R}^{dn \times dn} &\rightarrow \mathbb{R}^{dn \times dn} \\ \text{BlockDiag}_{d \times d}(\mathbf{M}) &\triangleq (\mathbf{M}_{11}, \dots, \mathbf{M}_{nn}) \end{aligned} \quad (6.19)$$

for the linear map that extracts the  $n$  diagonal blocks of a  $(d \times d)$ -block-structured matrix  $\mathbf{M}$ , we may expresss the orthogonal relaxation of (6.17) in an extrinsically-constrained form as

$$p_O^* = \min_{\mathbf{R} \in \mathbb{R}^{dn \times dn}} \text{tr}(\tilde{\mathbf{Q}} \mathbf{R}^\top \mathbf{R}) \quad \text{s.t.} \quad \text{BlockDiag}_{d \times d}(\mathbf{R}^\top \mathbf{R}) = (\mathbf{I}_d, \dots, \mathbf{I}_d). \quad (6.20)$$

Now we note that (6.20) has a structure similar to (QCQP), but where the vector  $\mathbf{x}$  is replaced with a matrix  $\mathbf{R}$ . While in (QCQP) we obtained a relaxation by replacing terms  $\mathbf{x}\mathbf{x}^\top$  with a suitable rank-1 matrix  $\mathbf{X}$  and dropping the rank-1

constraint, we can similarly replace the terms  $\mathbf{R}^\top \mathbf{R}$  in (6.20) with a rank-d matrix  $\mathbf{Z}$  and drop the rank constraint, thus obtaining the following semidefinite relaxation of the simplified PGO problem (6.17):

*Problem 3* (Semidefinite Relaxation for Pose-Graph Optimization)

$$p_{\text{SDP}}^* = \min_{\mathbf{Z} \in \mathbb{S}_+^{dn}} \text{tr}(\tilde{\mathbf{Q}} \mathbf{Z}) \quad \text{s.t.} \quad \text{BlockDiag}_{d \times d}(\mathbf{Z}) = (\mathbf{I}_d, \dots, \mathbf{I}_d). \quad (6.21)$$

As we saw in Section 6.1.1, this construction immediately implies that  $p_{\text{MLE}}^* \geq p_O^* \geq p_{\text{SDP}}^*$ . Moreover, if after solving the SDP relaxation (6.21), it so happens that the recovered minimizer  $\mathbf{Z}^* \in \mathbb{S}_+^{dn}$  admits a rank-d factorization of the form  $\mathbf{Z}^* = \mathbf{R}^{*\top} \mathbf{R}^*$  with  $\mathbf{R}^* \in \text{SO}(d)^n$ , then  $\mathbf{R}^*$  will itself be a *globally optimal* solution of the PGO Problem 2. The remarkable fact that justifies our interest in the relaxation (6.21) is that this favorable situation *actually occurs* in practice. Specifically, we have the following theorem (see [934] for a proof):

**Theorem 6.1** (Exact Recovery of PGO Solutions from Problem 3) *Let  $\tilde{\mathbf{Q}}$  be the matrix of the form (6.17b) constructed using the ground-truth relative transforms  $\tilde{\mathbf{T}}_{ij}$  in (6.8). There exists a constant  $\beta \triangleq \beta(\tilde{\mathbf{Q}}) > 0$  (depending upon  $\tilde{\mathbf{Q}}$ ) such that, if  $\|\tilde{\mathbf{Q}} - \bar{\mathbf{Q}}\|_2 < \beta$ , then:*

- 1 *The semidefinite relaxation Problem 3 has a unique solution  $\mathbf{Z}^*$ , and*
- 2  *$\mathbf{Z}^* = \mathbf{R}^{*\top} \mathbf{R}^*$ , where  $\mathbf{R}^* \in \text{SO}(d)^n$  is a minimizer of the maximum likelihood estimation Problem 2.*

In brief, Theorem 6.1 guarantees that as long as the noise corrupting the available measurements  $\tilde{\mathbf{T}}_{ij}$  is not too large, we can recover a *global minimizer*  $\mathbf{R}^*$  of Problem 2 (and thus also a global minimizer  $(\mathbf{R}^*, \mathbf{t}^*)$  of Problem 1 via (6.16)) by solving the SDP relaxation (6.21).

#### 6.1.2.3 Efficiently Solving the Relaxation via the Riemannian Staircase

As a semidefinite program, Problem 3 can *in principle* be solved efficiently, *i.e.*, in polynomial time. However, in practice the high computational cost of storing and manipulating the dense matrix decision variable  $\mathbf{Z}$  appearing in (6.21) prevents general-purpose interior-point methods from scaling effectively to problems in which the dimension of  $\mathbf{Z}$  is greater than a few thousand. Unfortunately, the instances of (6.21) arising in robotics and computer vision applications are typically one to two orders of magnitude larger than this maximum effective problem size, placing them well beyond the reach of general-purpose techniques. Consequently, in this subsection we develop a *specialized, structure-exploiting* optimization procedure that is capable of solving large-scale instances of Problem 3 efficiently.

**Exploiting Low-rank Structure.** The main idea behind our approach is to exploit the existence of *low-rank* solutions to Problem 3. Specifically, note that

while the *decision variable*  $\mathbf{Z}$  appearing in the relaxation (6.21) is a generic high-dimensional PSD matrix, Theorem 6.1 guarantees that the *solution*  $\mathbf{Z}^*$  we seek admits a very concise description in the factored form  $\mathbf{Z}^* = \mathbf{R}^{*\top} \mathbf{R}^*$  whenever exactness obtains. Moreover, it turns out that even when exactness fails to hold, minimizers of Problem 3 typically have a rank  $r$  not much greater than  $d$ , and therefore also admit a concise symmetric rank decomposition of the form  $\mathbf{Z}^* = \mathbf{Y}^{*\top} \mathbf{Y}^*$  for some  $\mathbf{Y}^* \in \mathbb{R}^{r \times dn}$ .

In their seminal work, Burer and Monteiro [125, 124] proposed an elegant general approach to exploit the existence of such low-rank solutions: simply replace every instance of the decision variable  $\mathbf{Z}$  in (6.21) with a *symmetric rank- $r$  factorization* of the form  $\mathbf{Y}^\top \mathbf{Y}$  (for some  $\mathbf{Y} \in \mathbb{R}^{r \times dn}$ ) to produce the following *Burer-Monteiro factorization* of (6.21):

$$p_{\text{SDPLR}}^*(r) = \min_{\mathbf{Y} \in \mathbb{R}^{r \times dn}} \text{tr}(\tilde{\mathbf{Q}} \mathbf{Y}^\top \mathbf{Y}) \quad \text{s.t.} \quad \text{BlockDiag}_{d \times d}(\mathbf{Y}^\top \mathbf{Y}) = (\mathbf{I}_d, \dots, \mathbf{I}_d). \quad (6.22)$$

We remark that since  $\mathbf{Y}^\top \mathbf{Y}$  is symmetric and PSD by construction, in (6.22) it is no longer necessary to explicitly enforce the positive-semidefiniteness constraint from the original SDP (6.21). Moreover, we note that this problem is strikingly similar to the problem we relaxed (6.20), with the important distinction that now the matrix  $\mathbf{Y}$  has size  $r \times dn$  instead of  $d \times dn$ , with  $r > d$ ; in other words, (6.22) reformulates the problem in a higher-dimensional space compared to (6.20).

If the maximum rank parameter  $r$  in (6.22) is chosen to be “small” (*i.e.*,  $r \ll dn$ ), then  $\dim(\mathbb{R}^{r \times dn}) = rnd \ll (dn + 1)dn/2 = \dim(\mathbb{S}_+^{dn})$ ; that is, the search space for (6.22) is *much* lower-dimensional than the search space for (6.21). Consequently, Burer and Monteiro proposed to apply fast nonlinear programming algorithms to the low-dimensional NLP (6.22) in order to search for a low-rank factor  $\mathbf{Y}^* \in \mathbb{R}^{r \times dn}$  of a minimizer  $\mathbf{Z}^* = \mathbf{Y}^{*\top} \mathbf{Y}^*$  of the original SDP (6.21).

**Exploiting Geometric Structure.** Note that if we additionally partition  $\mathbf{Y}$  into  $r \times d$  blocks as  $\mathbf{Y} = (\mathbf{Y}_1, \dots, \mathbf{Y}_n) \in \mathbb{R}^{r \times dn}$ , then the block-diagonal constraints appearing in (6.22) are equivalent to  $\mathbf{Y}_i^\top \mathbf{Y}_i = \mathbf{I}_d$  for all  $i \in [n]$ ; geometrically, this condition states that the columns of each block  $\mathbf{Y}_i \in \mathbb{R}^{r \times d}$  form an *orthonormal frame*. In general, the set of all orthonormal  $k$ -frames in  $\mathbb{R}^p$ ,

$$\text{St}(k, p) \triangleq \left\{ \mathbf{Y} \in \mathbb{R}^{p \times k} \mid \mathbf{Y}^\top \mathbf{Y} = \mathbf{I}_k \right\}, \quad (6.23)$$

forms a smooth compact matrix manifold, called the *Stiefel manifold*. This implies that the equality-constrained nonlinear program (6.22) is equivalent to the following *unconstrained* optimization problem defined on a product of Stiefel manifolds:

*Problem 4* (Burer-Monteiro-factored SDP relaxation as manifold optimization)

$$p_{\text{SDPLR}}^*(r) = \min_{\mathbf{Y} \in \text{St}(d, r)^n} \text{tr}(\tilde{\mathbf{Q}} \mathbf{Y}^\top \mathbf{Y}). \quad (6.24)$$

While formulations (6.22) and (6.24) are equivalent, the latter provides important

*computational* advantages. In particular, recognizing that the feasible set is a product of Stiefel manifolds enables us to apply specialized algorithms for optimization over smooth manifolds, which are substantially *simpler, faster, and more accurate* than general-purpose equality-constrained nonlinear programming techniques [106].

**Ensuring Global Optimality.** While the reduction from Problem 3 to Problem 4 dramatically reduces the size of the optimization problem that needs to be solved, it comes at the expense of (re)introducing the quadratic orthonormality constraints (6.23), which are nonconvex. It may therefore not be clear whether anything has really been gained by relaxing Problem 2 to Problem 4, since it appears that we may have simply replaced one difficult nonconvex optimization problem with another. The following remarkable result (adapted from Boumal et al. [107]) justifies this approach:

**Theorem 6.2** (A Sufficient Condition for Global Optimality in Problem 4) *If  $\mathbf{Y} \in \text{St}(d, r)^n$  is a (row) rank-deficient second-order critical point of Problem 4, then  $\mathbf{Y}$  is a global minimizer of Problem 4 and  $\mathbf{Z}^* = \mathbf{Y}^\top \mathbf{Y}$  is a solution of the semidefinite relaxation Problem 3.*

#### Riemannian Staircase

Theorem 6.2 immediately suggests a simple procedure, the *Riemannian Staircase*, for recovering solutions  $\mathbf{Z}^*$  of Problem 3 by applying fast *local* optimization algorithms to a *sequence* of instances of Problem 4. In brief, starting at some (small) initial maximum rank  $r \geq d$ , we apply a local solver to (6.24) (more precisely, a second-order Riemannian optimization algorithm) to recover a second-order critical point  $\mathbf{Y}^* \in \text{St}(d, r)^n$ . If  $\mathbf{Y}^*$  is rank-deficient, then Theorem 6.2 proves that  $\mathbf{Y}^*$  is a *global* minimizer of (6.24), and  $\mathbf{Z}^* = \mathbf{Y}^{*\top} \mathbf{Y}^*$  is a solution of (6.21). On the other hand, if  $\mathbf{Y}^*$  is *not* rank deficient, we can simply increase the maximum rank  $r$  and try again. Note that since *every*  $\mathbf{Y} \in \mathbb{R}^{r \times dn}$  is (row) rank-deficient for  $r \geq dn + 1$ , the Riemannian Staircase is *guaranteed* to terminate with an optimal solution  $\mathbf{Y}^*$  after finitely many iterations. However, typically only one or two “stairs” suffice.

Finally, we remark that from a practical standpoint, the Riemannian Staircase functions as a lightweight meta-algorithm that “wraps around” the same class of fast (second-order) *local* optimization algorithms commonly applied to SLAM problems in practice. This approach thus enables us to preserve the speed of current state-of-the-art SLAM techniques while *additionally* guaranteeing the recovery of *globally optimal* solutions, thereby achieving the best of both worlds.

#### 6.1.2.4 Rounding the Solution

We have just seen that the Riemannian Staircase provides an efficient algorithm for recovering a low-rank factor  $\mathbf{Y}^* \in \text{St}(d, r)^n$  of a solution  $\mathbf{Z}^* = \mathbf{Y}^{*\top} \mathbf{Y}^*$  of the relaxation Problem 3. However, ideally we would like to extract an optimal solution  $\mathbf{R}^* \in \text{SO}(d)^n$  of the PGO Problem 2 from  $\mathbf{Z}^*$  whenever the relaxation (6.21) is exact, and a *feasible approximate* solution  $\hat{\mathbf{R}} \in \text{SO}(d)^n$  otherwise. In this subsection, we describe an efficient rounding procedure that accomplishes these

**Algorithm 3** The SE-Sync algorithm

---

**Input:** An initial point  $\mathbf{Y} \in \text{St}(d, r_0)^n$ ,  $r_0 \geq d + 1$ .  
**Output:** A feasible estimate  $\hat{\mathbf{T}} \in \text{SE}(d)^n$  for the maximum likelihood estimation Problem 1, and the lower bound  $p_{\text{SDP}}^*$  for Problem 1's optimal value.

```

1: function SE-SYNC( $\mathbf{Y}$ )
2:   Set  $\mathbf{Y}^* \leftarrow \text{RIEMANNIANSTAIRCASE}(\mathbf{Y})$ .
3:   Set  $p_{\text{SDP}}^* \leftarrow F(\tilde{\mathbf{Q}}\mathbf{Y}^{*\top}\mathbf{Y}^*)$ .
4:   Set  $\hat{\mathbf{R}} \leftarrow \text{ROUNDSOLUTION}(\mathbf{Y}^*)$ .
5:   Recover the optimal translational estimates  $\hat{\mathbf{t}}$  corresponding to  $\hat{\mathbf{R}}$  via (6.16).
6:   Set  $\hat{\mathbf{T}} \leftarrow (\hat{\mathbf{t}}, \hat{\mathbf{R}})$ .
7:   return  $\{\hat{\mathbf{T}}, p_{\text{SDP}}^*\}$ 
8: end function
```

---

aims by operating directly on the low-rank factor  $\mathbf{Y}^*$  (*i.e.*, *without* the need to explicitly construct the dense, high-dimensional matrix  $\mathbf{Z}^*$ ).

The main insight that underpins our approach is that, in the event that the relaxation (6.21) is exact,  $\mathbf{R}^*$ ,  $\mathbf{Z}^*$ , and  $\mathbf{Y}^*$  will satisfy the relation

$$\mathbf{Y}^{*\top}\mathbf{Y}^* = \mathbf{Z}^* = \mathbf{R}^{*\top}\mathbf{R}^*. \quad (6.25)$$

In this case, equation (6.25) implies that the low-rank factor  $\mathbf{Y}^* \in \mathbb{R}^{r \times dn}$  actually has rank  $d$ , and consequently that  $\mathbf{R}^*$  can be recovered from  $\mathbf{Y}^*$  by computing a thin singular value decomposition of the latter. More generally, in the event that (6.21) is *not* exact, we can still recover an optimal *rank-d approximation*  $\hat{\mathbf{R}} \in \mathbb{R}^{d \times dn}$  of  $\mathbf{Y}^*$  using a *truncated* singular value decomposition, and then project the individual  $d \times d$  blocks  $\hat{\mathbf{R}}_i$  of  $\hat{\mathbf{R}}$  onto  $\text{SO}(d)$  (again using an SVD) to produce a *feasible approximate solution* of Problem 2.

#### 6.1.2.5 SE-Sync: The Complete Algorithm

Combining the efficient SDP optimization approach of Section 6.1.2.3 with the rounding procedure of Section 6.1.2.4 produces *SE-Sync* (Algorithm 3), our certifiably correct algorithm for pose-graph optimization [934].

When applied to an instance of PGO, SE-Sync returns a feasible point  $\hat{\mathbf{T}} \in \text{SE}(d)^n$  of the maximum likelihood estimation Problem 1 together with a lower bound  $p_{\text{SDP}}^* \leq p_{\text{MLE}}^*$  on its optimal value. This lower bound in turn provides an *upper* bound on the suboptimality of *any* feasible point  $\mathbf{T} = (\mathbf{t}, \mathbf{R}) \in \text{SE}(d)^n$  as a solution of Problem 1 according to

$$F(\tilde{\mathbf{Q}}\mathbf{R}^\top\mathbf{R}) - p_{\text{SDP}}^* \geq F(\tilde{\mathbf{Q}}\mathbf{R}^\top\mathbf{R}) - p_{\text{MLE}}^*. \quad (6.26)$$

Moreover, in the case that the relaxation in Problem 3 is exact, the estimate  $\hat{\mathbf{T}} =$

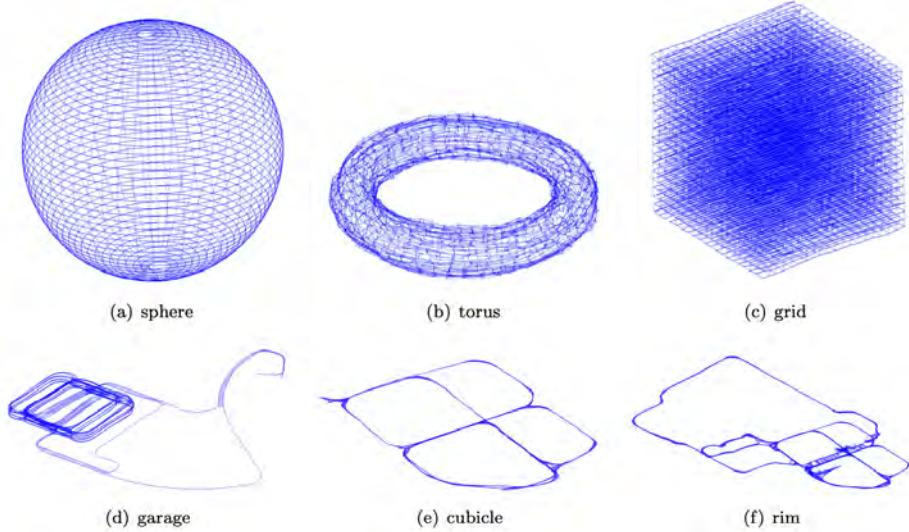


Figure 6.3 Globally optimal solutions for pose-graph optimization benchmarking datasets (From [934]).

$(\hat{\mathbf{t}}, \hat{\mathbf{R}}) \in \text{SE}(d)^n$  returned by Algorithm 3 *attains* this lower bound:

$$F(\tilde{\mathbf{Q}} \hat{\mathbf{R}}^T \hat{\mathbf{R}}) = p_{\text{SDP}}^*. \quad (6.27)$$

Consequently, verifying *a posteriori* that (6.27) holds provides a *computational certificate* of  $\hat{\mathbf{T}}$ 's correctness as a solution of Problem 1. SE-Sync is thus a *certifiably correct* algorithm for pose-graph optimization, as claimed.

Sample certifiably optimal results obtained with SE-Sync are shown in Figure 6.3 and discussed in depth in [934]. The paper [934] also reports a runtime analysis showing that the algorithm can be as fast if not faster than traditional local solvers.

### 6.1.3 Landmark-based SLAM

While in the previous section we have shown how to obtain a fast certifiable algorithm for PGO, in this section we show that the same derivation can be extended to landmark-based SLAM, specifically for the case where the robot takes bearing and range (*i.e.*, relative position) measurements to landmarks.

Let  $\mathbf{m}_{n+1}, \dots, \mathbf{m}_{n+\ell} \in \mathbb{R}^d$  be the set of  $\ell$  landmark positions that we wish to estimate in addition to the  $n$  robot poses. It turns out that these new landmark variables can be easily integrated into the existing implementation of SE-Sync. To do this, we treat the map point variables as *pure translation variables* (*i.e.*, poses without a rotational component).

We assume that we have a set of  $N_m$  landmark measurements,  $\{\tilde{\mathbf{m}}_{ik}\}$ , describing the relative position of landmark  $\mathbf{m}_k$  with respect to the  $i$ -th pose of the robot. We also assume that these measurements are corrupted by *additive zero-mean isotropic Gaussian noise* —the same form as the translation measurements in (6.8):

$$\tilde{\mathbf{m}}_{ik} = \bar{\mathbf{m}}_{ik} + \mathbf{m}_{ik}^\epsilon, \quad \mathbf{m}_{ik}^\epsilon \sim \mathcal{N}(0, \mu_{ij}^{-1} I_d). \quad (6.28)$$

To track these new measurements, we augment the pose-graph,  $\tilde{\mathcal{G}}$ , with a new set of vertices,  $\mathcal{V}_m$ , and edges,  $\tilde{\mathcal{E}}_m$ , with a one-to-one correspondence to the map variables and landmark measurements, respectively. Moreover, we denote with  $\tilde{\mathcal{E}}_r$  the edges corresponding to relative pose measurements of the robot pose (*e.g.*, odometry), which are assumed to follow the same measurement model as in (6.8). The maximum likelihood estimation problem given the landmark measurements and the relative pose measurements becomes:

*Problem 5* (Landmark-based SLAM)

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d, \mathbf{m}_k \in \mathbb{R}^d}} & \sum_{(i,j) \in \tilde{\mathcal{E}}_r} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,k) \in \tilde{\mathcal{E}}_m} \mu_{ik} \|\mathbf{m}_k - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{m}}_{ik}\|_2^2. \end{aligned} \quad (6.29)$$

It turns out that the remainder of the development we presented in the previous section can be used *as is*, if we treat the map variables as if they were pose translations. The optimal rotation variables can be found by applying the SE-Sync algorithm with a cost matrix  $\tilde{\mathbf{Q}}$  that has been modified to include the effect of the map variables. In particular, the weight matrix  $\boldsymbol{\Omega}$  and measurement matrix  $\tilde{\mathbf{D}}$  are redefined as follows:

$$\begin{aligned} \boldsymbol{\Omega} &= \text{BlockDiag}(\boldsymbol{\Omega}_\tau, \boldsymbol{\Omega}_\mu), \\ \boldsymbol{\Omega}_\tau &= \text{Diag}(\tau_1, \dots, \tau_{N_p}), \quad \tilde{\mathbf{D}}_{ek} = \begin{cases} -\tilde{\mathbf{t}}_{kj}^T, & e = (k, j) \in \tilde{\mathcal{E}}_r, \\ -\tilde{\mathbf{m}}_{kj}^T, & e = (k, j) \in \tilde{\mathcal{E}}_m, \\ \mathbf{0}_{1 \times d}, & \text{otherwise} \end{cases} \\ \boldsymbol{\Omega}_\mu &= \text{Diag}(\mu_1, \dots, \mu_{N_m}), \end{aligned} \quad (6.30)$$

Additionally, the projection matrix  $\boldsymbol{\Pi}$  is modified to account for the form of the new measurement graph,  $\tilde{\mathcal{G}} = (\mathcal{V} \cup \mathcal{V}_m, \tilde{\mathcal{E}}_r \cup \tilde{\mathcal{E}}_m)$ . Note that the connection Laplacian  $\mathbf{L}(\tilde{\mathcal{G}}^\rho)$  remains unaffected by the new map measurements.

Once the optimal rotations have been found, both the pose translations and the map points can be recovered in closed form similarly to (6.16):

$$[\mathbf{t}^{*T} \quad \mathbf{m}^{*T}]^T = -\text{vec}(\mathbf{R}^* \tilde{\mathbf{V}}^T \mathbf{L}(\mathcal{W}^\tau)^\dagger), \quad (6.31)$$

where we have assumed that the pose translation variables and map points have been ordered appropriately and  $\tilde{\mathbf{V}}$  and  $\mathbf{L}(\mathcal{W}^\tau)^\dagger$  have also been updated to include the map point measurements similarly to (6.30).

It is often the case in typical SLAM problems that there are many more map

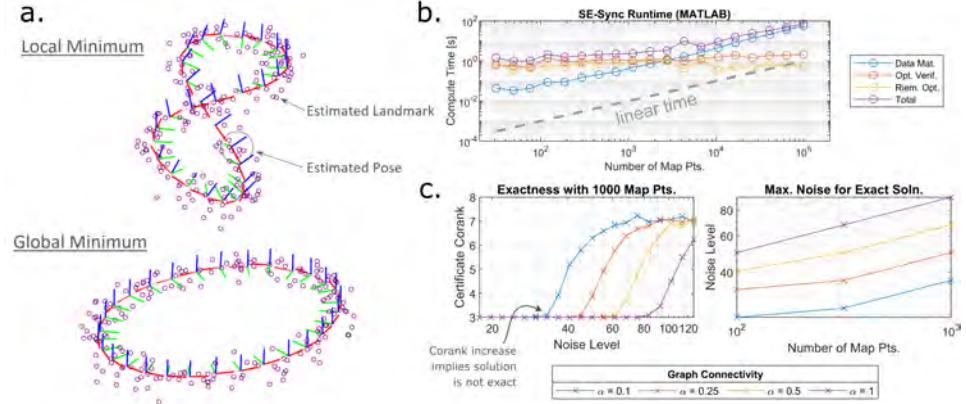


Figure 6.4 (a) An example of a local and global minimum for a simple landmark-based SLAM problem. (b) Runtime for the example shown in (a). Runtime increases linearly with respect to the size of the map and the bottleneck is the construction of the data matrix,  $\tilde{\mathbf{Q}}$  (shown in blue). (c) A study of the *exactness* of the relaxation via corank of a *certificate matrix* (exact when this metric is three). The noise on the measurements was set to a baseline standard deviation —0.866 meters for translation and 0.573 degrees for rotation— scaled by the “Noise Level” multiplier indicated in the plots. In general, the noise level for which the problem remains exact increases as the number of map points and the pose-to-map connectivity increase. From [465] (©2023 IEEE).

points than poses. As such, when including map variables in the SE-Sync formulation, it is important to ensure the algorithm remains efficient with respect to the number of map points in the problem. In the case where the number of map points is large, the bottleneck of the SE-Sync algorithm becomes the formation of the cost matrix  $\tilde{\mathbf{Q}}$ . It has been shown that the classical *Schur-complement trick* can be used to ensure that this matrix can be constructed with time complexity that is linear in the number of landmarks [465]. This linear dependence can be seen clearly in Figure 6.4(b), where it can also be seen that the computational cost of other components of SE-Sync algorithm does not increase with the number of landmarks.

Finally, the inclusion of the landmarks in the formulation has been shown to affect the exactness of the SDP relaxation. In particular, increasing the number of landmarks and the connectivity of the pose-to-map measurement graph have been shown to improve the exactness of the relaxation. More specifically, they increase the noise level for which a given problem has an exact relaxation. A demonstration of the effect of these parameters on a simple SLAM problem is shown in Figure 6.4(c).

### 6.1.4 Extensions: Range Measurements, Anisotropic Noise, and Outliers

This section shows that the machinery presented above (*i.e.*, Shor’s relaxation and the Riemannian Staircase solver) can be extended to other SLAM problems (Section 6.1.4.1). Moreover, we discuss more general tools to obtain semidefinite relaxations (which can be understood as a generalization of Shor’s relaxation) that further expand the set of certifiable algorithms for SLAM, but create additional challenges when solving the resulting SDP (Section 6.1.4.2).

#### 6.1.4.1 A Fast Certifiable Algorithm for Range-Aided SLAM

In the previous section, we showed how to develop certifiable algorithms for SLAM problems where the measurements are relative positions and relative rotations. Here we show that the same approach can be applied to problems involving range measurements, following the results presented in [837]. Specifically, we assume that we can measure distances  $\tilde{r}_{ij}$  between variables  $i$  and  $j$ , *e.g.*, the distance between two robot poses or between a landmark and a robot pose. The optimization problem for range-aided SLAM can be stated as follows.

*Problem 6* (Range-Aided SLAM)

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d}} \quad & \sum_{(i,j) \in \vec{\mathcal{E}}} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,j) \in \vec{\mathcal{E}}_d} \gamma_{ij} (\|\mathbf{t}_j - \mathbf{t}_i\| - \tilde{r}_{ij})^2 \end{aligned} \quad (6.32)$$

The first line in the objective function of Problem 6 is the same we have used in the previous sections: these are relative pose measurements, *e.g.*, corresponding to the robot odometry. The difference in this problem is the second line, which contains terms corresponding to the distance measurements  $\tilde{r}_{ij}$  (the set  $\vec{\mathcal{E}}_d$  is the set of pairs  $(i,j)$  such that a range measurement is available), weighted by the inverse variance  $\gamma_{ij}$  of these measurements. The range-only measurements  $\tilde{r}_{ij}$  are distinct in that they provide information about the relative distance between poses,  $i$  and  $j$ , but no information about the bearing or relative orientation. Problem 6 can be used to model a variety of practical SLAM problems, from landmark-based SLAM with distance measurements to landmarks, to multi-robot SLAM problems where the robots take relative range measurements.<sup>5</sup>

The challenge in applying Shor’s relaxation to Problem 6 is that (6.32) is *not* a QCQP: expanding the square  $(\|\mathbf{t}_j - \mathbf{t}_i\| - \tilde{r}_{ij})^2 = \|\mathbf{t}_j - \mathbf{t}_i\|^2 + 2\tilde{r}_{ij}\|\mathbf{t}_j - \mathbf{t}_i\| + \tilde{r}_{ij}^2$  reveals that this expression is *not* quadratic in the variables  $\mathbf{t}_i$  and  $\mathbf{t}_j$ , due to the unsquared norm term. To address this issue, the work [837] proposes an elegant

<sup>5</sup> Ultra-wideband (UWB) radios are popular sensors to obtain distance-only measurements.

reformulation of (6.32), which introduces auxiliary unit vectors  $\mathbf{b}_{ij}$  (with  $\|\mathbf{b}_{ij}\| = 1$ ):

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d \\ \mathbf{b}_{ij} \in S^{d-1}}} & \sum_{(i,j) \in \vec{\mathcal{E}}} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,j) \in \vec{\mathcal{E}}_d} \gamma_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \tilde{r}_{ij} \mathbf{b}_{ij}\|_2^2, \end{aligned} \quad (6.33)$$

Intuitively, formulation (6.33) also infers the *bearing*  $\mathbf{b}_{ij}$  between variables  $i$  and  $j$  for which a range measurement  $\tilde{r}_{ij}$  is available, and then recasts the corresponding term in the objective function, namely  $\|\mathbf{t}_j - \mathbf{t}_i - \tilde{r}_{ij} \mathbf{b}_{ij}\|^2$ , as a range-and-bearing measurement (similar to the ones we have seen in the previous sections). The advantage in doing so is that (6.33) is now a QCQP, since the last term in the objective is now quadratic in the unknowns, and the additional unit-norm constraints  $\|\mathbf{b}_{ij}\|^2 = 1$  on the  $\mathbf{b}_{ij}$  are quadratic as well. The new formulation can then be relaxed to an SDP using Shor's relaxation and solved efficiently using the Riemannian Staircase, leading to CORA [837], a fast certifiable algorithm for range-aided SLAM.

In contrast to the pose-and-landmark SLAM problems considered previously, the Shor relaxation of the range-aided SLAM problem (6.33) is not generically exact under bounded measurement noise. Specifically, the work [837] showed (empirically) that this relaxation is not typically exact in the multi-robot case unless there are relative pose measurements between the robots (*i.e.*, range-only measurements between the robots do not suffice to obtain exact relations). However, the relaxation is still exact in a variety of practical SLAM problems, and allows computing certifiably optimal solutions with runtime comparable to local solvers. More generally, Papalia et al. [837] show that the connectivity of the graph underlying the SLAM problem largely impacts the exactness of the SDP relaxation, a phenomenon also observed in [934, 465]. As we will see in Section 6.2, the graph connectivity also affects the accuracy of the SLAM estimate, creating very interesting graph-theoretic insights into the SLAM problem.

#### *6.1.4.2 Certifiable Algorithms Beyond Shor's Relaxation: Anisotropic Noise and Outliers*

So far we have reviewed certifiable algorithms (and fast solvers) for SLAM problems that can be reformulated as QCQPs. Below, we consider a broader class of SLAM problems, namely, problems with anisotropic measurement noise and problems with outliers. The interesting observation behind these problems is that, while strictly speaking they are no longer stated as QCQPs, they can often be formulated as Polynomial Optimization Problems (POPs), where both the objective and constraints are polynomial functions (instead of quadratic functions). This observation is important since there exists a generalization of Shor's relaxation, namely the *Moment (or Lasserre's) Relaxation*, that allows deriving SDP relaxations for POPs,

thus enabling the design of certifiable algorithms for a broader set of problems. Below, we review examples of SLAM problems that can be written as POPs, and then provide an overview of the Moment Relaxation and practical considerations.

**Example: Landmark-based SLAM with Anisotropic Noise.** In previous sections, we assumed the measurement noise to be isotropic. However, measurements produced by common sensing modalities (*e.g.*, stereo cameras, LiDAR, and radar) typically exhibit anisotropic noise. For instance, the measurement of the position of a landmark as observed by a stereo camera is typically more uncertain along the viewing direction of the cameras, due to the uncertainty induced by the stereo matching and triangulation process. More formally, the measurement model for the landmark measurements becomes  $\tilde{\mathbf{m}}_{ik} = \mathbf{R}_i^\top(\mathbf{m}_k - \mathbf{t}_i) + \boldsymbol{\epsilon}_{ik}$ , where  $\boldsymbol{\epsilon}_{ik} \sim \mathcal{N}(\mathbf{0}, \mathbf{W}_{ik})$  and  $\mathbf{W}_{ik}$  is an anisotropic covariance (*i.e.*,  $\mathbf{W}_{ik}$  cannot be written as a scalar multiple of the identity matrix). In the presence of anisotropic noise, we need to generalize the landmark-based SLAM Problem 5 as follows [465]:

*Problem 7* (Landmark-based SLAM with Anisotropic Noise)

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d, \mathbf{m}_k \in \mathbb{R}^d}} & \sum_{(i,j) \in \vec{\mathcal{E}}} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,k) \in \vec{\mathcal{E}}_m} \|\mathbf{R}_i^\top(\mathbf{m}_k - \mathbf{t}_i) - \tilde{\mathbf{m}}_{ik}\|_{\mathbf{W}_{ik}}^2 \end{aligned} \quad (6.34)$$

In (6.34),  $\vec{\mathcal{E}}_m$  denotes the set of edges corresponding to landmark measurements, and, for a generic vector  $\mathbf{a}$  and matrix  $\mathbf{W}$  of suitable dimensions, the notation  $\|\mathbf{a}\|_{\mathbf{W}}^2 = \mathbf{a}^\top \mathbf{W} \mathbf{a}$  denotes the standard Mahalanobis squared norm. This seemingly innocuous change with respect to Problem 5 is quite consequential in practice. Indeed, if  $\mathbf{W}_{ik}$  is isotropic (*e.g.*,  $\mathbf{W}_{ik} = \mu_{ik}^{-1} \mathbf{I}_3$ ) we can manipulate the expression in (6.34) to the QCQP in Problem 5, whereas if  $\mathbf{W}_{ik}$  is anisotropic the problem is *quartic* (*i.e.*, it involves degree 4 polynomials in the variables).<sup>6</sup>

**Example: SLAM with Outliers.** So far, we assumed all measurements to be affected by zero-mean (but possibly anisotropic) Gaussian noise. Unfortunately, as we discussed in Chapter 3, in real SLAM problem some measurements might be outliers. This is typically the case for loop closure or landmark measurements, where incorrect place recognition or data association might cause adding incorrect measurements to the SLAM back-end. As we discussed in Chapter 3, an effective approach to mitigate the impact of outliers is to use robust loss functions. For instance, in the presence of outliers, the landmark-based SLAM Problem 5 becomes:

<sup>6</sup> Intuitively, when  $\mathbf{W}_{ik} = \mu_{ik}^{-1} \mathbf{I}_3$ , we can simplify the quartic function  $\|\mathbf{R}_i^\top(\mathbf{m}_k - \mathbf{t}_i) - \tilde{\mathbf{m}}_{ik}\|_{\mathbf{W}_{ik}}^2$  to a quadratic one:  $\|\mathbf{R}_i^\top(\mathbf{m}_k - \mathbf{t}_i) - \tilde{\mathbf{m}}_{ik}\|_{\mathbf{W}_{ik}}^2 = \mu_{ik} \|\mathbf{R}_i^\top(\mathbf{m}_k - \mathbf{t}_i) - \tilde{\mathbf{m}}_{ik}\|_2^2 = \mu_{ik} \|\mathbf{m}_k - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{m}}_{ik}\|_2^2$ , where in the last equality we used the rotation invariance of the  $\ell_2$  norm.

*Problem 8* (Pose-Graph Optimization with Outliers)

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d}} & \sum_{(i,j) \in \vec{\mathcal{E}}_o} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,k) \in \vec{\mathcal{E}}_m} \rho(\sqrt{\mu_{ik}} \|\mathbf{m}_k - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{m}}_{ik}\|_2). \end{aligned} \quad (6.35)$$

In (6.35),  $\rho(\cdot)$  is a robust loss function (*cf.* Chapter 3) which is designed to reduce the influence of potential outliers in the landmark measurements. From Chapter 3, we know that we can use the Black-Rangarajan duality to reformulate (6.35) as a least-squares problem with auxiliary variables  $w_{ik}$ , one for each robust loss term. For instance, when choosing  $\rho(\cdot)$  to be the truncated quadratic loss, then (6.35) can be rewritten as

$$\begin{aligned} \min_{\substack{\mathbf{R}_i \in \text{SO}(d) \\ \mathbf{t}_i \in \mathbb{R}^d \\ w_{ik} \in [0,1]}} & \sum_{(i,j) \in \vec{\mathcal{E}}_o} \kappa_{ij} \|\mathbf{R}_j - \mathbf{R}_i \tilde{\mathbf{R}}_{ij}\|_F^2 + \tau_{ij} \|\mathbf{t}_j - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{t}}_{ij}\|_2^2 \\ & + \sum_{(i,k) \in \vec{\mathcal{E}}_m} w_{ik} \mu_{ik} \|\mathbf{m}_k - \mathbf{t}_i - \mathbf{R}_i \tilde{\mathbf{m}}_{ik}\|_2^2 + (1 - w_{ik}) \beta^2, \end{aligned} \quad (6.36)$$

where  $\beta$  is the maximum inlier error, as specified by the truncated quadratic loss. We note that (6.36) now includes auxiliary variables  $w_{ik}$ , which indicate whether a measurement is classified as an inlier or outlier. Interestingly, the objective function in (6.36) includes polynomials of degree up to 3, while the constraints are still at most quadratic functions. The work [1218] shows that the same conclusion holds for several choices of robust losses and for other variations of the problem, including pose-graph optimization and multiple rotation averaging. Below, we discuss how to obtain semidefinite relaxations for optimization problems involving polynomials.

**Polynomial Optimization Problems and Moment Relaxation.** The previous examples show how a broad range of SLAM problems can be reformulated as optimization problems involving polynomials. More formally, they can be written as POPs:

$$\begin{aligned} \min_{\mathbf{x}} & \quad p(\mathbf{x}) \\ \text{subject to } & h_i(\mathbf{x}) = 0, \quad i = 1, \dots, n_h \\ & g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, n_g, \end{aligned} \quad (\text{POP})$$

where the functions  $p, h_i, g_i$  are real polynomials in the variable  $\mathbf{x}$ , and  $n_h$  and  $n_g$  are the number of equality and inequality constraints.

Rewriting the SLAM problem as (POP) does not immediately imply any computational advantage: POPs are a very general class of optimization problems (which also includes QCQPs) and are generally intractable to solve. Our interest towards (POP) stems from the fact that there exists a standard procedure,

known as the *moment (or Lasserre's) relaxation*, to obtain a semidefinite relaxation of (POP). Even more interestingly, the procedure provides tools to obtain a *hierarchy of relaxations* and also guarantees that —under mild assumptions— certain relaxations in this hierarchy are exact. While we refer the reader to [147] and the seminal works [625, 624] for a more extensive introduction to the moment relaxation, below we provide a simple example to convey the underlying ideas.

To illustrate how to obtain a relaxation of a POP, consider the following problem:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & p(\boldsymbol{x}) \\ \text{subject to } & h_i(\boldsymbol{x}) = 0, \quad i = 1, \dots, n_h \end{aligned} \tag{6.37}$$

where, for simplicity, we assume  $\boldsymbol{x} = [x_1; x_2]$ , and that  $p(\boldsymbol{x})$  and  $h(\boldsymbol{x})$  are polynomials of degree at most 4; in this simplified example we only have equality constraints. Then, to derive the moment relaxation of (POP), we define the vector of monomials of degree up to  $r = 2$ , where  $r$  is called the *order* of the relaxation:

$$[\boldsymbol{x}]_2 = [1; x_1; x_2; x_1^2; x_1 x_2; x_2^2]. \tag{6.38}$$

Then, we can form the moment matrix as the following outer product:

$$\boldsymbol{X}_4 \triangleq [\boldsymbol{x}]_2 [\boldsymbol{x}]_2^\top = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 \\ x_1 & x_1^2 & x_1 x_2 & x_1^3 & x_1^2 x_2 & x_1 x_2^2 \\ x_2 & x_1 x_2 & x_2^2 & x_1^2 x_2 & x_1 x_2^2 & x_2^3 \\ x_1^2 & x_1^3 & x_1^2 x_2 & x_1^4 & x_1^3 x_2 & x_1^2 x_2^2 \\ x_1 x_2 & x_1^2 x_2 & x_1 x_2^2 & x_1^3 x_2 & x_1^2 x_2^2 & x_1 x_2^3 \\ x_2^2 & x_1 x_2^2 & x_2^3 & x_1^2 x_2^2 & x_1 x_2^3 & x_2^4 \end{bmatrix}. \tag{6.39}$$

Now the key observation is that we can write any polynomial of degree up to 4 as a linear combination of the entries of  $\boldsymbol{X}_4$ . Therefore, we can rewrite (6.37) as:

$$\begin{aligned} \min_{\boldsymbol{X}, \boldsymbol{x}} \quad & \text{tr}(\boldsymbol{C}\boldsymbol{X}) \\ \text{subject to } & \text{tr}(\boldsymbol{H}_i \boldsymbol{X}) = 0, \quad i = 1, \dots, n_h \\ & \boldsymbol{X} = [\boldsymbol{x}]_2 [\boldsymbol{x}]_2^\top \end{aligned} \tag{6.40}$$

In full analogy with Shor's relaxation, we can now replace the constraint  $\boldsymbol{X} = [\boldsymbol{x}]_2 [\boldsymbol{x}]_2^\top$  with  $\boldsymbol{X} \succeq 0$  and  $\text{rank}(\boldsymbol{X}) = 1$  and then relax the rank constraint to obtain a semidefinite relaxation. Importantly, the moment relaxation also adds *redundant constraints*<sup>7</sup> to improve its quality. These constraints capture the fact that the moment matrix contains repeated entries (*e.g.*, the term  $x_1 x_2$  appears 3 times in (6.40)) as well as the fact that if  $h_i(\boldsymbol{x}) = 0$ , then also  $x_1 \cdot h_i(\boldsymbol{x}) = 0$  and  $x_2 \cdot h_i(\boldsymbol{x}) = 0$  must hold. The moment relaxation provides a systematic way to identify all these redundant constraints, and also extends to the case of inequality constraints. Moreover, while above we derived an order  $r = 2$  relaxation, we can repeat the procedure for any integer  $r \geq 2$ , obtaining larger but better SDP

<sup>7</sup> These are constraints that are inconsequential for problem (6.40), but improve the quality of the relaxation.

relaxations (hence a *hierarchy* of relaxations). Indeed, the seminal work [803] establishes that the moment relaxation will produce exact relaxations (*i.e.*, recovers a certifiably optimal solution to the original POP) at a finite order  $r$ , under mild assumptions. Interestingly, related work [1218, 465] has observed that moment relaxations of problems with anisotropic noise and outliers are already exact at a low relaxation order. Moreover, they remain tight even when using a subset of the variables in the monomial basis (6.38) (*i.e.*, a *sparse* monomial basis), which further reduces the size of the resulting SDP.

**The ‘Catch’: Solving the SDP Moment Relaxation.** One important difference between the relaxations we have seen in the previous sections and the ones discussed in this section is that the latter involve a large number of redundant constraints in the SDP relaxation. While this difference might seem minor, the consequences for the SDP solver are profound. In particular, SDPs with redundant constraints (as the ones typically obtained from the moment relaxation) are *degenerate* [27], which makes the application of the Riemannian Staircase solver problematic for two reasons. First, constraint qualification conditions typically fail for these problems, and these conditions are required to ensure convergence of the Riemannian Staircase [838]. Second, degenerate SDPs have an infinite number of dual solutions, which creates computational obstacles in the implementation of the Riemannian Staircase. In other words, the Riemannian Staircase is no longer a viable solver for the degenerate SDPs typically produced by the moment relaxation. While the recent literature includes specialized solvers for moment relaxations of POPs [1217], these solvers are still relatively slow compared to local solvers.

## 6.2 How Accurate is the Optimal Solution of a SLAM Problem?

In the previous section we discussed how to obtain certifiably optimal solutions to certain SLAM problems. However, another fundamental question remains: *how accurate is the optimal solution compared to the ground truth?* Understanding this fundamental limit and identifying the key factors that influence estimation accuracy provides critical guidance for both system designers and end users. For example, at design time such insights can guide the choice of sensors the robot is equipped with; on the other hand, at deployment time, these insights can be used to guide the motion of the robot (and the corresponding acquisition of measurements) to ensure robust SLAM performance. Before offering a concrete answer, we first need to formalize the question. Modern SLAM pipelines are complex, with many interacting subsystems affecting overall performance. Consequently, we approach this question from an estimation-theoretic perspective, focusing on the SLAM back-end.

**Some Terminology and Facts.** The goal of the SLAM back-end is to estimate unknown quantities such as robot poses and landmarks’ positions from noisy measurements. These measurements are random variables because they are corrupted by random sensor noise. An estimator is a function that maps collected noisy mea-

surements to an estimate of the unknown parameters. Since the estimator depends on random measurements, it is itself a random variable. The Mean Squared Error (MSE) is a commonly used metric for evaluating an estimator's performance. As the name suggests, the MSE represents the squared estimation error averaged over all possible measurements. In the univariate case, the MSE can be expressed as the sum of the estimator's squared bias and its variance. The bias refers to the difference between the ground truth and the estimator's output, averaged over all possible measurements. The variance captures the variability of the estimator's output around its mean. The relationship between bias and variance, and the definition of the MSE extends naturally to the multivariate setting, where the variance is replaced by a covariance matrix. The MSE is still the sum of the squared norm of the bias and the trace of the covariance matrix. Clearly, for *unbiased* estimators (*i.e.*, estimators with zero bias), the only quantity influencing the MSE is the covariance matrix. In the following, we show how to lower bound the covariance of the SLAM estimate, thus getting a fundamental limit on the accuracy achievable by a SLAM system.

### 6.2.1 Cramér-Rao Lower Bound and the Fisher Information Matrix

The *Cramér-Rao Lower Bound (CRLB)* provides a theoretical limit on the best *Cramér-Rao lower bound* estimator covariance achievable by any unbiased estimator. Formally,

$$\text{Cov}(\hat{\mathbf{x}}) \succeq \mathcal{I}(\mathbf{x}_{\text{true}})^{-1}, \quad (6.41)$$

where  $\hat{\mathbf{x}}$  is any unbiased estimator of  $\mathbf{x}_{\text{true}} \in \mathbb{R}^m$ , and  $\mathbf{A} \succeq \mathbf{B}$  indicates that  $\mathbf{A} - \mathbf{B}$  is positive semidefinite. The matrix  $\mathcal{I}(\mathbf{x}_{\text{true}})$  appearing on the right-hand side of (6.41) is the *Fisher information matrix (FIM)*, which is defined as<sup>8</sup>

$$[\mathcal{I}(\mathbf{x}_{\text{true}})]_{i,j} \triangleq \mathbb{E}_{\mathbf{z}} \left[ \frac{\partial}{\partial x_i} \log p(\mathbf{z}; \mathbf{x}) \frac{\partial}{\partial x_j} \log p(\mathbf{z}; \mathbf{x}) \right]. \quad (6.42)$$

*Fisher information matrix*

Here the expectation is taken over the possible realizations of measurements  $\mathbf{z}$  drawn from the probability density function  $p(\mathbf{z}; \mathbf{x}_{\text{true}})$ , and the partial derivatives of the log-likelihood function are *evaluated at the true parameter value*  $\mathbf{x}_{\text{true}}$ . Under certain regularity conditions, the FIM can also be expressed as the expected value of the Hessian of the log-likelihood:

$$[\mathcal{I}(\mathbf{x}_{\text{true}})]_{i,j} = -\mathbb{E}_{\mathbf{z}} \left[ \frac{\partial^2}{\partial x_i \partial x_j} \log p(\mathbf{z}; \mathbf{x}) \right]. \quad (6.43)$$

<sup>8</sup> The classical CRLB assumes that the variables live in a Euclidean space. In SLAM, we often need to estimate poses and rotation matrices. The CRLB has been extended to Riemannian manifolds and matrix Lie group parameters [105, 95]. In such cases, CRLB has an additional (additive) term that depends on the curvature of the parameter space. The curvature term is often negligible when the signal-to-noise ratio is large.

This connection to the Hessian provides an intuitive understanding of the CRLB: the CRLB establishes a lower bound on the covariance of any unbiased estimator, expressed in terms of the local sensitivity (curvature) of the (expected) log-likelihood function with respect to the parameters around the true parameter value  $\mathbf{x}_{\text{true}}$ . If the log-likelihood is relatively flat around the true parameter value across different realizations of measurements (*i.e.*, in expectation), any unbiased estimator will struggle to accurately localize the true parameter based on the observed data. In such cases, unbiased estimators will exhibit higher variance, leading to a higher MSE. In summary, the FIM captures the amount of information one can learn about the true parameter's value from measurements using any unbiased estimator.

Under certain regularity conditions, it has been shown that the maximum likelihood estimator  $\hat{\mathbf{x}}_{\text{mle}}$  asymptotically (*i.e.*, when the number of measurements tends to  $\infty$ ) converges (in distribution) to  $\mathcal{N}(\mathbf{x}_{\text{true}}, \mathcal{I}(\mathbf{x}_{\text{true}})^{-1})$ . Therefore, the maximum likelihood estimator is *asymptotically* unbiased and achieves the CRLB (*i.e.*, minimum variance among all unbiased estimators). Since the true value of parameters  $\mathbf{x}_{\text{true}}$  is unknown, we often approximate the FIM by  $\mathcal{I}(\hat{\mathbf{x}}_{\text{mle}})$ . Furthermore, it is common to approximate the covariance of the maximum likelihood estimator  $\hat{\mathbf{x}}_{\text{mle}}$  with  $\mathcal{I}(\hat{\mathbf{x}}_{\text{mle}})^{-1}$ .

As an example, consider the common scenario where measurements are generated by corrupting a smooth (potentially nonlinear) function with additive Gaussian noise. In this case, the measurement model can be written as

$$\mathbf{z} = \mathbf{h}(\mathbf{x}_{\text{true}}) + \boldsymbol{\epsilon}, \quad (6.44)$$

where  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \Sigma)$  is the noise. The likelihood function *evaluated at*  $\mathbf{x}$  is thus given by  $p(\mathbf{z}; \mathbf{x}) = \mathcal{N}(\mathbf{h}(\mathbf{x}), \Sigma)$ . Plugging this likelihood in (6.42) yields the FIM (evaluated at  $\mathbf{x}$ ):

$$\mathcal{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \Sigma^{-1} \mathbf{J}(\mathbf{x}), \quad (6.45)$$

in which  $\mathbf{J}(\mathbf{x})$  denotes the Jacobian of the measurement model  $\mathbf{h}$  evaluated at  $\mathbf{x}$ .<sup>9</sup>

**A Scalar Measure of Uncertainty: Optimal Experimental Design Criteria.** In most applications, we would like our estimates to be as certain and accurate as possible. The FIM quantifies certainty, but it is generally matrix-valued; ideally we want a single value that quantifies uncertainty that we can use to make decisions or evaluate our estimator. Therefore, in many applications (*e.g.*, system design, active SLAM), one needs to map the FIM, a PSD matrix, to a real number that captures a meaningful and ‘optimizable’ aspect of the estimation error uncertainty. Standard choices have been investigated in the field of Optimal Experimental Design. These include the determinant of the FIM (D-optimality), the trace of its inverse (A-optimality), and its smallest eigenvalue (E-optimality). These criteria are *spectral* functions of the FIM (*i.e.*, can be computed from the eigenvalues of the

<sup>9</sup> The matrix (6.45) is often called the *information matrix* in the SLAM literature.

FIM). Each criterion reflects a different aspect of estimation error: the D-optimality criterion quantifies the uncertainty hyper-volume, the A-optimality criterion measures the average variance, and the E-optimality criterion represents the worst-case estimation variance.

### 6.2.2 Fisher Information Matrix and Graph Laplacian

In this section, we study how the graphical structure of SLAM problems affects the FIM. Inspecting the FIM (6.45) reveals the intuitive fact that the measurement noise covariance matrix  $\Sigma$  affects the FIM: unsurprisingly, higher measurement noise covariance increases the lower bound on the smallest achievable MSE among all unbiased estimators in the CRLB. Additionally, the Jacobians of the measurements appear in (6.45), but it is difficult to intuitively understand from this equation *how* the measurements' Jacobian affects the CRLB.<sup>10</sup> In the following, we gain insights on the structure of the Jacobians and the resulting FIM by relating them to properties of the graph underlying the SLAM problem.

All variants of SLAM naturally admit a graphical representation as we have seen in Chapter 1 and earlier in this chapter. The graph essentially encodes “who is observing what” and provides a concise overview of the SLAM problem. For instance, in pose-graph optimization and landmark-based SLAM, each variable (e.g., robot pose or landmark position) is represented by a vertex, while pairwise measurements (pose-pose or pose-landmark) correspond to edges between the respective vertices.

Next, we study how properties of the graph underlying the SLAM problem impact the accuracy of the resulting estimate. In particular, the degree of *connectivity* within the graph reflects the *redundancy* in measurements. Intuitively, a “better” connected SLAM graph is expected to be more robust to noise, yielding accurate estimates even under higher noise levels due to redundant measurements. While it is straightforward to show from (6.45) that introducing additional measurements (*i.e.*, edges) always reduces the lower bound in the CRLB (in the Loewner order),<sup>11</sup> the impact of different measurements varies depending on which variables are involved in the additional measurements (*i.e.*, the resulting graph connectivity). This is particularly evident in the context of loop closure in SLAM: closing a “larger” loop has a more significant effect on improving the accuracy of the SLAM solution. This intuition has been formalized in a series of works, establishing connections between the graphical structure of SLAM and desirable properties in both estimation and optimization. Below we provide a brief overview of these findings.

#### Connections between FIM and Graph Laplacian for a Simplified PGO

<sup>10</sup> Note that even without understanding the structure of the Jacobians, we can still compute the FIM (approximated at the current estimate) and use it to assess the estimation accuracy of a given SLAM problem. However, a deeper understanding of the FIM will allow us to *predict* how modifications of the SLAM problem (*e.g.*, driven by active data collection by the robot) will impact the future uncertainty in the SLAM estimates, thus informing active perception and design tasks.

<sup>11</sup> Each new measurement adds a positive semi-definite matrix to the FIM. This is known as the “information never hurts” principle.

**Problem.** The FIM in landmark-based SLAM and pose-graph optimization is closely connected to the graph Laplacian [560, 559, 874, 190]. This relationship is intuitive given that the measurements in these frameworks consist of pairwise relative observations between vertices. To illustrate the concept, we derive the FIM for the simpler problem of estimating robot positions in a 3D pose-graph optimization problem when the robot orientations are known. Let  $\mathbf{z}_k$  denote the  $k$ -th relative measurement in which pose  $i_k$  observes pose  $j_k$  in its local frame:

$$\mathbf{z}_k = \mathbf{R}_{i_k}^\top (\mathbf{t}_{j_k} - \mathbf{t}_{i_k}) + \boldsymbol{\epsilon}_k, \quad (6.46)$$

where  $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, w_k^{-1} \mathbf{I}_3)$  in which  $\mathbf{I}_3$  is the  $3 \times 3$  identity matrix. Let  $\mathbf{z}$ ,  $\mathbf{t}$ , and  $\boldsymbol{\epsilon}$  be the stacked vectors of measurements, positions, and noise variables. Additionally, let  $\mathbf{R}$  be the block-diagonal matrix of (known) rotation matrices such that the  $k$ -th block is the rotation matrix involved in the  $k$ -th measurement:

$$\mathbf{R} \triangleq \text{BlockDiag}(\mathbf{R}_{i_1}, \mathbf{R}_{i_2}, \dots, \mathbf{R}_{i_m}). \quad (6.47)$$

The stacked measurement model can be expressed as:

$$\mathbf{z} = \mathbf{R}^\top (\mathbf{A} \otimes \mathbf{I}_3)^\top \mathbf{t} + \boldsymbol{\epsilon}, \quad (6.48)$$

where  $\mathbf{A}$  denotes the reduced incidence matrix of the pose graph,<sup>12</sup> and  $\otimes$  denotes the Kronecker product. Therefore, the Jacobian matrix of the stacked measurement model is given by

$$\mathbf{J} = \mathbf{R}^\top (\mathbf{A} \otimes \mathbf{I}_3)^\top. \quad (6.49)$$

The information matrix of the stacked noise vector is given by

$$\boldsymbol{\Sigma}^{-1} = \text{BlockDiag}(w_1 \mathbf{I}_3, w_2 \mathbf{I}_3, \dots, w_m \mathbf{I}_3). \quad (6.50)$$

Let  $\mathbf{W}$  denote the diagonal matrix of edge weights:

$$\mathbf{W} \triangleq \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix}. \quad (6.51)$$

Using (6.45), the FIM can be computed as follows:

$$\mathcal{I} = \mathbf{J}^\top \boldsymbol{\Sigma}^{-1} \mathbf{J} \quad (6.52a)$$

$$= (\mathbf{A} \otimes \mathbf{I}_3) \mathbf{R} \boldsymbol{\Sigma}^{-1} \mathbf{R}^\top (\mathbf{A} \otimes \mathbf{I}_3)^\top \quad (6.52b)$$

$$= (\mathbf{A} \otimes \mathbf{I}_3) (\mathbf{W} \otimes \mathbf{I}_3) (\mathbf{A} \otimes \mathbf{I}_3)^\top \quad (6.52c)$$

$$= (\mathbf{A} \mathbf{W} \mathbf{A}^\top) \otimes \mathbf{I}_3 \quad (6.52d)$$

$$= \mathbf{L}_w \otimes \mathbf{I}_3, \quad (6.52e)$$

<sup>12</sup> The *reduced* incidence matrix is obtained by removing the row(s) corresponding to anchored poses from the incidence matrix [148]; in SLAM, it is typical to anchor the first pose to be the origin of the world frame. This is done to remove gauge symmetry due to the symmetric nature of SLAM with relative measurements.

where we observed that the block diagonal entries of  $\Sigma^{-1}$  are invariant to rotation, hence  $\mathbf{R}\Sigma^{-1}\mathbf{R}^\top = \Sigma^{-1}$ , and we used the fact that  $\Sigma^{-1} = \mathbf{W} \otimes \mathbf{I}_3$ . In (6.52a),  $\mathbf{L}_w = \mathbf{A}\mathbf{W}\mathbf{A}^\top$  is the reduced weighted Laplacian matrix of the graph where edge weights are given by  $w_1, w_2, \dots, w_m$ . This clearly shows that in this case (simplified PGO where robot orientations are assumed known and noise is assumed isotropic), the FIM is fully characterized by the reduced weighted Laplacian of the underlying graph.

**FIM for Other SLAM Problems.** The result above can be generalized to PGO and landmark-based SLAM problems [560, 874, 190]. In these problems, the FIM involves both the reduced weighted Laplacian matrix of graph and additional terms that depend on the robot trajectory. In particular, the FIM for 3D pose-graph optimization can be written as [874, Eq. 29]:

$$\mathcal{I}(\mathbf{x}) = \sum_{k=1}^m \mathbf{L}_k \otimes (\text{Ad}(\mathbf{T}_{i_k}^{-1})^\top \Sigma_k^{-1} \text{Ad}(\mathbf{T}_{i_k}^{-1})) , \quad (6.53)$$

where  $\mathbf{T}_{i_k}$  is the pose of the robot making the  $k$ -th measurement,  $\Sigma_k$  is the covariance matrix of the noise corrupting the  $k$ -th measurement, and  $\mathbf{L}_k$  is the reduced *elementary* Laplacian matrix of the  $k$ -th edge defined as  $\mathbf{L}_k \triangleq \mathbf{a}_k \mathbf{a}_k^\top$  where  $\mathbf{a}_k$  is the  $k$ -th column of the reduced incidence matrix of the graph. Empirical results and theoretical analyses demonstrate that, under certain conditions, the (approximate) optimal design criteria derived from the reduced Laplacian closely match those obtained from the FIM [560, 874, 190].

**Practical Considerations.** The connection between the FIM and the graph Laplacian allows approximating optimal design criteria—which are spectral functions of the FIM—using the spectrum of the reduced weighted Laplacian. For instance, the D-optimality criterion can be approximated by a function of the determinant of the weighted reduced Laplacian. According to Kirchhoff’s matrix-tree theorem, this determinant equals the weighted number of spanning trees of the graph [560]. Since the weighted number of spanning trees serves as a measure of connectivity in edge-weighted graphs, this result formalizes the intuition that a graph’s connectivity directly influences estimation accuracy. Similarly, the E-optimality criterion is related to the graph’s algebraic connectivity [275, 560, 874]. Overall, measuring uncertainty using the graph Laplacian instead of the FIM offers two main advantages: it leads to more computationally efficient techniques (*e.g.*, the Laplacian has a dimension six times smaller than that of the FIM in 3D pose-graph optimization), and it eliminates the need to solve the SLAM problem or acquire actual measurements, as the calculations are based solely on the graph structure. These graphical approximations of design criteria have been successfully applied in active SLAM [190, 581, 874]—where a robot plans its trajectory to maximize the anticipated SLAM accuracy—and in measurement selection and pruning [275, 560, 1091], where the goal is to select and retain only the *most informative* measurements in lifelong SLAM problems.

active SLAM

### 6.3 Further Readings & Recent Trends

**Certifiable Algorithms.** The first certifiable algorithm for 2D SLAM traces back to [149], and builds on earlier work done in computer vision and related areas, including [338, 1011]. Extensions to 3D SLAM and variations quickly followed [151, 1105, 934, 936, 114, 152], with SE-Sync providing the first blueprint for building *fast* certifiable algorithms, as well as the first guarantees of global optimality (*i.e.*, exactness of the relaxation) under bounded measurement noise [934]. Since then, certifiable algorithms have been designed for a variety of problems related to SLAM,<sup>13</sup> including rotation averaging [122, 969, 251], PGO [934, 149], landmark-based SLAM [465], multi-robot SLAM [1090], range-aided SLAM [837], 3D registration [115, 1220], multi-set registration [501], 2-view geometry [116, 1282, 367, 1094, 537], perspective-n-point problems [1050], calibration [382], single-frame pose and shape estimation [1003], multi-frame pose and shape estimation [991], and structure from motion with learned depth [1253]. Recent work has also extended certifiable algorithms to cope with anisotropic noise [466] and outliers [1218].

While there has been incredible progress in this area over the last decade, three exciting open questions remain. First of all, there are still SLAM problems that cannot be attacked with certifiable algorithms. For instance, when doing visual SLAM (Chapter 7), the perspective projection arising in the objective function is a rational function rather than a polynomial; while the problem can still be reformulated as a POP by introducing extra variables, it is typically impractical to add one variable for each keypoint measurement in an image. Similarly, the modeling of IMU measurements (that we discuss in detail in Chapter 11) has not been conducive to the design of certifiable algorithms. Second, except for certain problems, *e.g.*, [934, 537, 1253, 837], where the Riemannian Staircase method can be applied as a fast solver, SDPs might need to be solved by interior-point methods or other ad hoc solvers. While interior-point methods are very effective and fast in solving low-dimensional optimization problems arising in the SLAM front-end (*e.g.*, [115, 1220, 116, 1282, 367, 1094, 537]), they become impractically slow when applied to large-scale problems in the SLAM back-end (*e.g.*, [934]), and ad hoc solvers —while being more scalable— are still relatively slow compared to local solvers [1217]. Recent work to address this problem has not only investigated faster SDP solvers, but has also focused on how to reduce the size of the SDPs (*e.g.*, by sparsifying the monomial basis underlying the moment relaxation) [1217], or how to reduce the number of constraints, in the attempt to make the SDPs non-degenerate or faster to solve [290]. Finally, while current works compute per-instance certificates of optimality (*i.e.*, they compute an estimate and possibly provide a certificate of optimality for it), the literature lacks a fundamental understanding of *when* the

<sup>13</sup> Some of these optimization problems are solved as part of the SLAM front-end, *e.g.*, to get relative poses from sensor measurements.

relaxation is expected to be exact, with only few papers providing *conditions for exactness* in specific problems [1215, 856, 934, 310, 147].

**Problems with Outliers.** Real-world SLAM problems are typically plagued with outliers, a problem that we discussed at length in Chapter 3. While several works wrap outlier-free certifiable algorithms in a Graduated Non-Convexity outer loop [1220, 1003, 991] to gain empirical robustness to outliers, recent work directly attempts to develop certifiable algorithms for robust estimation problems involving outliers. Efforts in this direction tackled PGO [616, 153], rotation estimation [1215], 3D registration [1220], multiple rotation averaging, absolute pose estimation, and pose and shape estimation [1218]. A good summary of these results is provided in [1218], while connections with parallel work in robust statistics are discussed in [147]. These algorithms rely on the moment relaxation described earlier in this chapter and the resulting SDPs are still relatively slow to solve. An alternative approach has been to obtain solutions via local solvers, and use the same insights behind the moment relaxation to derive methods to only *check* optimality [1216].

Many interesting open questions also remain in this case.<sup>14</sup> In the presence of outliers, we still see the same challenges and opportunities discussed in the previous paragraph, including how to extend certifiable algorithms to other problems with outliers, how to design faster solvers, and how to derive conditions for exactness (*e.g.*, as a function of the amount of noise and outliers). At the same time, there are additional challenges related to the presence of outliers. First of all, the SLAM approaches discussed above assume an outlier-free odometry backbone: in other words, only the loop closure measurements are wrapped in a robust loss function. The assumption of having a reliable odometry source is acceptable in many SLAM problems, but can be restrictive in certain cases. For instance, in visual SLAM problems, the odometry can become unreliable if the feature tracking fails, while in multi-robot SLAM problems, there is no odometry backbone connecting the different robots and all the inter-robot measurements can be outliers. While the formulation of current certifiable algorithms would extend to the case where also the odometry is wrapped in a robust loss, the resulting relaxations are known to be loose, *e.g.*, [616], and it is unclear how to improve them. Second, even if we compute an optimal estimate when solving a robust estimation problem, if the majority of the measurements are outliers, the estimate can still be grossly incorrect. An interesting and relatively unexplored area is to design certifiable algorithms that can recover *multiple* hypotheses while guaranteeing that at least one hypothesis is correct, a setting called *list decodable regression* in statistics [147].

**Uncertainty Quantification and Downstream Applications.** While Section 6.2 provided computational tools to bound the uncertainty in our SLAM estimate and potentially predict its evolution as a function of the structure of the underlying graph, it still leaves many questions open. First of all, the computation of

<sup>14</sup> The reader is also referred to [147] to a more in-depth discussion of some of these challenges.

the covariance of the SLAM estimate relies upon the knowledge of the measurement covariances: if the measurement covariance matrices are inaccurate, the resulting uncertainty bounds become unreliable. Recent work uses learning to estimate the measurement covariances or even the entire measurement model [1017, 886, 1241] (this is also related to our discussion of differentiable optimization in Chapter 4). Second, the traditional approach to compute a covariance for the estimate does not account for the potential presence of outliers. In the presence of outliers, the estimated covariance can be incorrect and the distribution of potential estimates (*e.g.*, of the robot trajectory) can become highly multimodal, hence limiting the use of the covariance estimates described earlier in this chapter. Finally, there is a growing literature using uncertainty quantification to guide active perception and active SLAM (see [873] for a survey) and to determine how to subselect measurements during life-long SLAM or in the presence of resource constraints [275, 560, 1091, 150].

## **PART TWO**

### SLAM IN PRACTICE



## II

### Prelude

Ayoung Kim, Timothy Barfoot, Luca Carlone,  
Frank Dellaert, and Daniel Cremers

Part I laid the foundations of SLAM by introducing the basic language of factor graphs and their role as a SLAM back-end. Building on this groundwork, Part II focuses on the characteristics and integration of various sensor modalities used in SLAM, which directly impacts the SLAM front-end. Together, these two parts provide a cohesive understanding of the SLAM pipeline: the back-end, discussed in Part I, addresses the underlying optimization problem, while the front-end, covered in this part, tackles sensor-specific tasks such as pre-processing, time synchronization, noise filtering, and measurement modeling.

This part explores widely adopted sensors in SLAM, and the corresponding chapters are organized—based on the common categorization of SLAM algorithms—by their primary sensor type. The sensors discussed in Part II include RGB cameras and LiDARs, along with emerging modalities such as event cameras and radars. IMUs are emphasized for their critical role in SLAM, both as standalone sensors and in combination with other modalities. The discussion also explores how robot kinematics can be integrated as measurements in the SLAM system.

#### II.1 Key Modules in the SLAM Front-End

Most SLAM front-ends rely on two key modules: odometry and loop closure. While these modules produce essential measurements, priors can be occasionally incorporated alongside them. The *odometry* module produces measurement between consecutive nodes in a graph, typically arranged in sequential time order. By chaining together these odometry measurements, a trajectory can be inferred, though it will inevitably accumulate drift over time. The accumulated drift highlights the importance of the *loop closure* module as a key component in the SLAM system, as it mitigates this drift by recognizing previously visited scenes and establishing connections to historical nodes. Incorporating loop closure detection and correction is the essence of SLAM. Beyond these two key modules, additional sensor measurements may also be included. For instance, one can add unary factors modeling additional sensor data or priors (*e.g.*, GPS or pose priors), enhancing the overall SLAM system’s accuracy and robustness.

*odometry****II.1.1 Odometry***

The odometry module estimates the relative transformation between two consecutive nodes in a factor graph. This transformation can vary in complexity depending on the scenario. The most common case involves two nodes corresponding to consecutive sensor frames, where a relative 6-DOF binary factor is inferred between them. In this context, the comparison of sensor measurements is performed by computing pixel-to-pixel, point-to-point, or feature-to-feature matching. Many visual and LiDAR SLAM systems compute odometry in this manner.

At times, the odometry computation does not rely on matching consecutive measurements. When kinematic or dynamic information is available, such as from radar or leg encoders and contact sensors, the odometry can be directly computed by velocity integration. For instance, in radar, both range and radial velocity can be measured. By using the radial velocity to infer ego-velocity, a 6-DOF transformation can be integrated between two frames. Leg odometry can be obtained by incorporating data from encoders or contact sensors, along with robot kinematics. One of the most widely used odometry methods is inertial odometry. Inertial measurements involve more complex computations between two nodes, often implemented as a preintegration factor. This preintegration can serve as odometry but is frequently combined with other sensors to reduce the odometric drift.

The odometry estimation module is typically named after the primary sensor used for estimation, *e.g.*, visual odometry, LiDAR odometry, visual-inertial odometry, leg odometry, and so on. In this part, each chapter provides a detailed exploration of the odometry module tailored to specific sensor modalities.

*loop closure****II.1.2 Loop Closure***

The loop closure module addresses two problems: *loop closure detection* and *relative pose estimation*. The first problem, which is also referred to as *place recognition*, involves using information-retrieval methods to identify a candidate loop closure in a topological manner. This task involves identifying a query's nearest index from a database, using retrieval or matching algorithms. In SLAM, the query is the current sensor measurement and the database includes a collection of previously observed places in the form of raw sensor measurements or descriptors.<sup>1</sup> Then the goal is to find a place in the database that matches the current observation, hence detecting the case where the robot is revisiting a previously explored area.

Loop closure detection is commonly performed using a variety of exteroceptive sensors, such as RGB cameras, event cameras, LiDARs, and radars. Early work focused on creating highly descriptive and compact place (or scene) descriptors. For instance, visual place recognition applied information retrieval techniques, introduc-

<sup>1</sup> We use the terms *query* and *database*, which are commonly used in the information-retrieval literature, but also commonly used in place recognition.

ing visual words to implement a bag-of-words model. Primarily, binary bag-of-words (DBoW) [362] has been widely adopted in many visual SLAM applications. Similarly, for range sensors, compact descriptors for range measurements, such as those used in Scan Context [562], have been developed. These hand-crafted descriptors are now transitioning to learning-based approaches, including cross-modal place recognition, which enables place recognition from different sensor modalities.

When detecting a loop closure, both discernibility and scalability are crucial. Place recognition must accurately distinguish between similar-looking but distinct locations, mitigating perceptual aliasing. To achieve this, it is essential to develop effective descriptors or train networks that ensure strong discernibility. In addition, during long-term SLAM operations over large areas, both the map and the query database will grow. Consequently, loop closure detection must be performed efficiently, even as the map expands.

Once a candidate is found, *re-localization* or *relative pose estimation* aims to estimate a fine registration between the candidate match and the current data, which is needed for the SLAM back-end optimization. Once the proposal loop closure candidate is detected, the follow-up registration estimates a relative transformation (either full 6-DOF or partial) between the query node and the candidate match node, typically resulting in a binary factor.

### II.1.3 Priors and Unary Factors

Some sensors measure properties of a single node in the factor graph rather than providing relative measurements between nodes. This is the case for sensors like GPS in terrestrial navigation, depth sensors in underwater environments, or radar providing instantaneous velocity measurements. These measurements can often be incorporated into the SLAM system as unary factors, constraining a single node.

The factor graph might also include priors, for instance enforcing the first node to be at the origin of the world frame, or potentially enforcing desired motion properties (*e.g.*, zero-velocity at certain nodes). Unary factors and priors will not be extensively discussed in the following chapters, and readers are referred to dedicated resources on GNSS [145] or UWB applications.

## II.2 Sensors and Factor Graphs

From the factor-graph SLAM perspective, each sensing modality produces a type of factor, acting as a building block for the entire factor graph. An example factor graph is given in Figure II.1. The factor-graph framework, introduced in Part I, serves as a generic optimization back-end, where each sensor can contribute to the graph either independently or by being combined with other sensors.

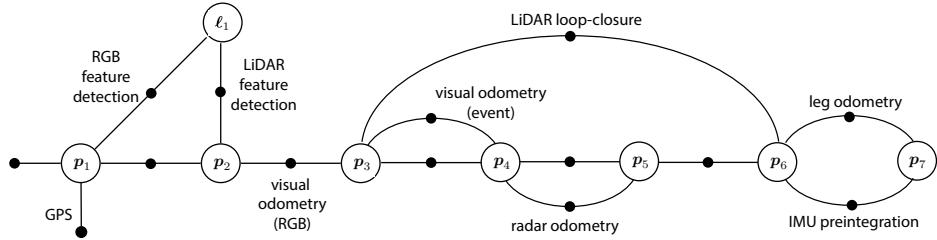


Figure II.1 Sample factor-graph including all sensing modalities introduced in this part. Here the variable nodes  $p_i$  are generic states (*e.g.*, poses, velocities, *etc.*) at each time  $i$ .

### II.2.1 Selecting the Right Sensor for Your Application

Let us briefly overview exteroceptive sensors commonly used in SLAM systems, highlighting their advantages and limitations.

**RGB Camera:** RGB cameras are among the most widely adopted sensors in SLAM applications due to their affordability, ability to capture semantic information and appearance, and their resemblance to the human visual system. These cameras capture rich visual context, including semantic information, which serves as a powerful cue for various visual SLAM tasks.

However, they do have notable limitations. One significant drawback is their reliance on adequate lighting conditions; without sufficient illumination, the quality of captured images degrades, which can hinder performance in low-light environments. Additionally, RGB cameras are sensitive to glare, reflections, motion blur, and shadows, and their performance is significantly limited in extreme environments, such as those with fog or smoke.

**LiDAR:** LiDARs directly measure distance without the need for triangulation, enabling to accurately capture 2D/3D geometry from range measurements. Indeed, range sensors, such as 2D LiDAR and sonar, are foundational sensing modalities in the history of SLAM. The point cloud data generated by LiDAR has been widely adopted for dense 2D/3D mapping in both indoor and outdoor environments, making it especially valuable for large-scale applications in construction sites, urban areas, and forests.

However, LiDARs remain expensive in terms of cost, memory and power consumption, as well as downstream computational requirements. In particular, 3D LiDARs generate large point clouds at each scans, requiring substantial computational power and memory for processing and storage. Moreover, 3D LiDAR sensors are still bulky, and size and weight requirements become a consideration, as compared to smaller sensors, such as RGB cameras. These challenges should be carefully considered when using LiDAR in budget-sensitive, real-time systems (*e.g.*, drones).

Furthermore, despite their robustness to illumination conditions, their performance can be impaired in adverse weather conditions, such as heavy rain, fog, or snow.

**Radar:** Radar is less affected by environmental conditions, making it a reliable sensor for deployment in extreme environments. It can provide velocity information through Doppler measurements, adding an additional layer of functionality. While the localization and mapping accuracy of radar is often lower than that of LiDAR, it remains one of the few robust sensors capable of enabling SLAM in extreme environments where LiDAR and cameras may be impaired. However, radar data tends to be noisy and extremely sparse, which can complicate its processing. The signal processing required for radar data is also computationally intensive, and the sensor measurements are not as intuitive as those from optical sensors, making them more challenging to interpret and integrate into systems.

**Event camera:** Event cameras are unique sensors that offer extremely high temporal resolution (in the order of microseconds) at a low power consumption. By transmitting data only for pixels that change, they are highly energy-efficient, and capture fast-moving objects without motion blur. Their asynchronous nature also leads to efficient data processing, generating smaller data streams compared to conventional frame-based cameras that operate at the same sampling rate. These features make event cameras ideal for real-time, high-speed applications. Additionally, their ability to handle both very bright and dark conditions with high dynamic range makes them more effective than frame-based cameras in challenging lighting environments. Despite their advantages, their main limitations are their noise, and lack of fine texture details and absolute intensity output.

### II.2.2 Sensor Fusion

In the SLAM literature, it is common practice to combine multiple complementary sensors to obtain high-performing systems. Inertial measurement units (IMUs) are often used due to their pervasiveness and low-SWAP nature. Incorporating an IMU with other sensors significantly enhances the performance of many SLAM systems. IMUs provide valuable information about motion, velocity, and orientation, which improves the accuracy and robustness of systems like visual-inertial odometry, LiDAR-inertial odometry, and radar-inertial odometry. These systems leverage the preintegration of IMU data to maintain continuous tracking and reduce drift over time. While IMU-only systems have seen some development, they remain limited in terms of accuracy and reliability. When combined with other sensors like LiDAR or cameras, however, IMUs strengthen the overall SLAM system, compensating for the weaknesses of individual sensors and enabling more precise and reliable mapping and localization, especially in dynamic or challenging environments.

Incorporating the kinematics of the robot platform is also crucial. Many SLAM systems in robotics are designed to work with specific platforms, such as ground

robots, drones, or legged robots. By formulating the kinematics of these platforms and integrating them with other sensor modalities, the performance and robustness of the system can be significantly enhanced.

As we mentioned above, sensor fusion can further benefit from the complementarity and heterogeneity of the sensors used in the SLAM system. Camera and LiDAR systems are one of the most popular combinations. This is because semantic information from camera and direct range measurement from LiDAR can effectively compensate for each sensor's limitation. Camera-radar fusion has similar benefits, complementing each other to mutually enhance performance and address limitations. For instance, the velocity measurements from radar can be used to detect dynamic objects instantaneously, while the semantic information from the RGB camera adds valuable context. Similarly, the combination of LiDAR and radar is beneficial, as radar can improve odometry and facilitate dynamic object removal, while the dense point clouds from LiDAR contribute to higher-quality mapping and submap matching.

Lastly, there are often heterogeneous characteristics even among sensors of the same type. RGB cameras, for instance, can vary significantly depending on factors such as lens type and shutter mechanism. When combining multiple LiDARs, the beam pattern, point cloud density, and field of view (FOV) can vary significantly between different LiDAR models, and this discrepancy must be addressed when integrating LiDAR with other sensors. This is also true for radar, where the two main types —spinning radar and SoC radar— differ significantly in terms of data type, measurement techniques, and their applications.

### ***II.2.3 Calibration and Synchronization of Sensors***

*calibration*

Sensor calibration can be categorized into two types: intrinsic and extrinsic. Intrinsic calibration focuses on determining the model parameters specific to a sensor, such as the focal length and distortion coefficients for a camera or the intensity calibration for LiDAR. In the case of a camera using a pinhole camera model, intrinsic calibration estimates parameters like focal length and distortion coefficients. Intrinsic calibration begins with a sensor model and solves for the parameters that define that model.

On the other hand, extrinsic calibration involves finding the transformation between multiple sensors, aligning their coordinate systems, and ensuring accurate data fusion. This process includes establishing correspondences and using them in an optimization problem to solve for the relative transformation between two sensors. The core challenge arises when establishing correspondences across different modalities. For example, to match a pixel from an RGB camera to a 3D point from a LiDAR, one needs to solve a multi-modal registration problem. Due to differences in data formats and underlying physics, comparing data from two sensors is often challenging.

Both intrinsic and extrinsic are typically performed in the factory, before the actual robot deployment. A widely adopted solution is to use a target to carefully capture data from both sensors in order to solve the registration problem. However, this calibration is often limited to the target and may suffer from drift over time. To address this, targetless calibration methods have been developed, which leverage the surroundings as calibration features. Alternatively, calibration can be updated adaptively during deployment by including the calibration parameters in the SLAM factor-graph optimization formulation.

When integrating multiple sensors into a SLAM system, time synchronization must be carefully managed. Each sensor operates at its own sampling rate, meaning data from different sensors arrives at different intervals. In robotics, the movement of the platform further exacerbates this issue, as the motion induces discrepancies between sensor data streams. Proper synchronization ensures that all sensor data is aligned in time, enabling accurate fusion and minimizing errors in mapping and localization. Of course, strict hardware synchronization is not always feasible, and interpolation —potentially using faster sampling sensors like an IMU— may be necessary to bridge the gap between sensor data streams.

*synchronization*

### II.3 Evaluation

SLAM evaluation is typically conducted from multiple perspectives, often at the level of individual modules such as odometry, place recognition, and mapping. As expected, real-time performance is crucial for odometry, whereas place recognition and mapping can tolerate slower processing rates. More computationally intensive tasks, such as map maintenance and updates, are often deferred to post-processing.

*evaluation*

The most common SLAM evaluation metric is the trajectory accuracy, typically measured by comparing the estimated path to a ground-truth trajectory. However, generating ground truth requires expensive sensors and often extensive post-processing. While this trajectory-to-trajectory comparison sounds straightforward, it is not always feasible —especially in environments like indoors, where RTK GPS is ineffective. In such cases, a few surveyed points using QR markers or artificial targets can serve as an alternative. Furthermore, SLAM performance can be analyzed from multiple perspectives by focusing on different quantitative metrics [373, 1280, 405] during trajectory evaluation.

For place recognition, standard classification metrics such as the precision-recall curve, AUC, and F1 score are commonly used. In the context of robot re-localization, however, not only the accuracy metrics but also the distribution [567] of candidate matches plays a critical role in assessing performance.

Lastly, map evaluation is often the most challenging aspect of the SLAM evaluation, not only due to the large spatial scale but also because of the difficulty in obtaining accurate ground truth. Static LiDAR systems, such as terrestrial laser scanners (TLS), are commonly used to create high-fidelity reference maps for com-

parison. In addition to global accuracy, evaluations should also consider fine-grained structural details and the memory efficiency of the map representation.

#### **II.4 How to Read Part II?**

The chapters in this part are organized by sensing modality. Each chapter follows a similar structure, focusing on odometry, place recognition, and overall SLAM system for each sensor. It is recommended to start with the visual SLAM chapter to grasp the basic definitions of each SLAM module.

Following the discussion on visual SLAM in Chapter 7, two range sensors are introduced: LiDAR SLAM in Chapter 8 and radar SLAM in Chapter 9. Moving beyond conventional sensors, we delve into event camera SLAM in Chapter 10. Inertial measurement units (IMUs) are covered in Chapter 11. Additionally, this part explores how to model odometry for legged-robot systems in Chapter 12.

# 7

## Visual SLAM

Jakob Engel, Juan D. Tardós, Javier Civera, Margarita Chli,  
Stefan Leutenegger, Frank Dellaert, and Daniel Cremers

Reconstructing the world and the sensor motion from cameras is a challenge referred to as Visual Simultaneous Localization and Mapping, shortened to Visual SLAM or VSLAM. With cameras being omni-present, inexpensive, and power-efficient, the potential of visual SLAM for autonomous robots, self-driving cars, or mixed and augmented reality is endless. In this chapter, we first provide some historical background and terminology (Section 7.1), and give an overview of a typical visual SLAM pipeline (Section 7.2). Then, we review key ingredients of visual SLAM formulations (Section 7.3), and cover more advanced topics in image alignment and bundle adjustment (BA) (Section 7.4). Afterwards, we describe examples of visual SLAM systems (Chapter 7.5), real-time 3D reconstruction (Chapter 7.6), SLAM with depth-cameras (Chapter 7.7), and discuss the advantages of combining vision with other sensing modalities, including IMU, GPS, and WiFi (Chapter 7.8). We close the chapter with a brief discussion about new trends (Chapter 7.9).

*Visual SLAM*  
*VSLAM*

### 7.1 Historical Background and Terminology

#### 7.1.1 From Photogrammetry to Bundle Adjustment and Visual SLAM

Visual SLAM’s history builds upon general SLAM methods, but it is also rooted in advances from the photogrammetry and computer vision communities. We highlight here the historical connection with these two last fields, the connection with SLAM being already addressed throughout the handbook.

In 1822, Nicéphore Niépce invented modern photography with the oldest surviving photograph being “A View from a Window at Le Gras” in 1827. The history of reconstructing the world from cameras started a few decades after the invention of photography. The French army officer Aimé Laussedat is often considered the inventor of photogrammetry as he pioneered the use of terrestrial photographs for topographic mapping around 1851. In 1867 the German engineer Albrecht Meydenbauer further developed photogrammetry for architectural surveys. From 1890 onward the German mathematician and mountaineer Sebastian Finsterwalder (president of the German Mathematical Society from 1915 onward) pioneered the use

*bundle adjustment*

of aerial imagery for photogrammetric reconstruction of glaciers in the Alps and also advocated techniques of projective geometry [357]. His doctoral student Otto von Gruber formalized the mathematical framework of bundle adjustment (BA) for reconstruction of structure and motion from a set of corresponding points observed in multiple images. These concepts were developed around the beginning of the 20th century, well before the advent of computers. The deployment on computers was later pioneered by Hellmut H. Schmid, a German rocket scientist who developed matrix computation techniques for BA and teamed up with American Duane C. Brown in the 1950s to deploy these methods on the largest computers of their time.

*structure from motion*

Reconstruction methods and the field of structure from motion (SFM) research build on various camera models starting from the pinhole camera model and the use of projective geometry to capture the relationship between 2D point observations and their corresponding 3D world coordinates. In the early 1990s Tomasi and Kanade [1099] developed matrix factorization techniques for the reconstruction of static scenes under the simplified assumption of an orthographic projection. Whereas earlier approaches often focused on the reconstruction from two views, in the 2000s the community shifted to the problem of multi-view structure from motion. Traditionally, the reconstruction pipeline involved feature extraction, correspondence estimation, the use of minimal solutions for obtaining an initial camera configuration and a subsequent BA to obtain a globally consistent reconstruction. Much effort was therefore dedicated to the development of feature extraction and matching algorithms with feature descriptors such as SIFT [696] or SURF [62], and more recently a multitude of learning-based descriptors. In order to cope with incorrect point correspondences researchers developed sampling strategies such as RANSAC [328] that allowed the method to revisit the correspondence estimation in alternation with model fitting (*cf.* Chapter 3).

Whereas structure from motion is often focused on the accurate (generally off-line) reconstruction of large-scale 3D worlds from an unordered collection of images, visual SLAM typically focuses on the online and real-time reconstruction from a moving camera. A prerequisite for such online and real-time approaches was therefore the development of causal methods such as [209] which focus on the challenge of optimal structure from motion given only the past images (as opposed to the entire image collection or video). The first real-time capable methods for structure from motion / visual SLAM emerged around 2000 [523, 244].

### 7.1.2 Terminology

The following terms are often used interchangeably to describe similar processes; however, they place different emphasis depending on application and community they are used in.

*photogrammetry*

**Photogrammetry** is the science of extracting accurate measurements, spatial

information, and 3D reconstructions from 2D photographs. By analyzing overlapping images taken from different viewpoints, photogrammetry enables the creation of geometric representations of objects or environments. This technique is widely used in mapping, surveying, and 3D modeling, forming the foundation for many visual odometry and SLAM algorithms.

**Bundle adjustment (BA)** is a mathematical optimization method used to refine 3D reconstructions. It adjusts the positions, orientations, and optionally intrinsic parameters of cameras, along with the 3D positions of observed points, to minimize the *reprojection error* —the difference between observed image points and the points projected from the 3D model. Optimization is typically performed using second-order methods such as Gauss-Newton or Levenberg-Marquardt and requires careful initialization and robust outlier rejection to converge effectively. Recent research has also explored *initialization-free BA*, which aims to simplify the optimization process.

**Structure from motion (SFM)** refers to the process of reconstructing 3D structures from a collection of 2D images taken from different perspectives. Unlike photogrammetry, which often assumes known camera positions, SFM simultaneously estimates both the 3D structure of the scene and the motion of the cameras. SFM is typically applied in non-causal, non-real-time scenarios, where images may come from diverse sources (*e.g.*, internet photo collections) rather than a continuous video stream. Typically in the final stage SfM will resort to BA for optimization. Landmark projects, such as *Building Rome in a Day* [14], demonstrate its scalability and potential for large-scale applications.

**Visual odometry (VO)** focuses on estimating the motion of a camera by analyzing sequential visual frames in a video. It identifies visual correspondences between consecutive images to measure the relative motion of the sensor over time. VO primarily deals with local motion estimation and operates within a sliding window of recent observations, without building a global map. However, VO is often combined with a loop closure detection and mapping component to form a complete visual SLAM system.

**Visual SLAM** is a computational technique that enables a system to simultaneously localize itself within an unknown environment and build a map of that environment in real time. It combines elements of photogrammetry, visual odometry, and structure from motion to process image data, track camera motion, and construct detailed 3D maps. In contrast to VO, it will typically employ an explicit functionality of recognizing previously visited places, re-localize relative to them, and optionally adjust pose estimates around such a “loop” —a process referred to as *loop closure*. Visual SLAM is a cornerstone technology for robotics, autonomous vehicles, and augmented reality, where precise navigation and environmental understanding are essential.

## 7.2 The Processing Pipeline of a Visual SLAM System

Building a complete visual-SLAM system involves combining various components into a cohesive framework that can handle the demands of real-time operation, scalability, and robustness. Key considerations include deciding when and how each component operates, structuring the compute and data-flow efficiently, and ensuring adaptability to diverse environments. As in LiDAR SLAM (*cf.* introductory discussion in Chapter I and Chapter 8), the problem of visual SLAM can be tackled into multiple stages (*i.e.*, by splitting computation into a SLAM front-end and a SLAM back-end). In contrast to LiDAR-based systems, however, the 3D geometry is not directly measured since one rather observes projections of the scene irradiance onto the screen. That makes the overall estimation in visual-SLAM problems more challenging. Modern complete Visual-SLAM systems typically include three core sub-functionalities that complement each other: an odometry front-end, a mapping back-end, and a loop closure and re-localization component. In this section, we mostly provide an overview of the pipeline, while we postpone a more detailed description to Section 7.3.

### 7.2.1 Visual Odometry Front-End

The core element of a visual-SLAM system is visual odometry, which aims at estimating relative motion between consecutive camera frames. This stage provides the initial estimate for the camera's pose. As discussed above, there are two alternative approaches to compute visual odometry: (i) Feature-based approaches split the challenge into three stages of detecting and extracting feature points, computing pairwise correspondence across images and subsequently determine the relative camera motion by minimizing the re-projection error with respect to camera motion and 3D point coordinates. The last stage is quite analogous to classical BA. (ii) Direct approaches tackle the problem in one step where a photometric loss function is directly optimized with respect to camera motion and 3D structure. They are therefore quite related to approaches of optical flow and what is sometimes called photometric BA.

At least in their naive, first formulations, feature-based methods have demonstrated a larger basin of convergence thanks to explicit data associations. To alleviate this, direct methods thus often employ a coarse-to-fine approach, *i.e.*, start with aligning down-sampled images, or even attempt to align dense (learned) features instead of brightness or color.

### 7.2.2 Mapping Back-End

The back-end optimizes the trajectory and map using global optimization techniques like BA or pose-graph optimization. This step refines the estimates provided

by the front-end and integrates observations (including the ones resulting from place recognition) into a consistent map. As a consequence, one obtains more long-term consistency, and long-range distortions are reduced.

### 7.2.3 Visual Place Recognition and Relocalization

Visual odometry is prone to drift because errors in the camera tracking will accumulate over time. In the absence of absolute positioning sensors, like GPS, one can eliminate drift and enforce global consistency by aligning the current image to previously observed images. To this end, one needs to compute correspondence across a potentially large set of images. This can be done either by an efficient matching of classical feature descriptors like SIFT, SURF or BRIEF —or by means of suitable trained neural networks, an approach that has become increasingly popular in the last years. The resulting component detects when the camera revisits a previously mapped area (loop closure detection), correcting accumulated drift and re-establishing localization when tracking fails.

### 7.2.4 Compute and Data Flow

Efficient data flow is essential for a well-performing SLAM system:

**Pipeline Parallelism.** Different components, such as tracking, mapping, and optimization, often run in parallel to maximize efficiency.

**Data Sharing.** Intermediate outputs, like keypoints or poses, are shared between components to minimize redundant computation.

**Adaptive Scheduling.** Compute-heavy tasks, like global optimization, are scheduled based on system requirements, prioritizing real-time responsiveness.

## 7.3 Visual SLAM Fundamentals

Let us now review the ingredients involved in the implementation of the visual-SLAM front-end, back-end, and visual place recognition.

### 7.3.1 Camera Model

A parameterized description of the sensor that models image formation from the observed scene should include a **geometric component** (also called the projection function), which describes how 3D points are mapped to 2D pixels, and a **photometric component**, which describes how physical light intensity (radiance) maps to pixel values.



Figure 7.1 A central requirement for visual SLAM systems is the choice of a suitable lens. Shown here are BF2M2020S23 ( $195^\circ$ ), BF5M13720 ( $183^\circ$ ), BM4018S118 ( $126^\circ$ ), BM2820 ( $122^\circ$ ), and a GoPro replacement lens ( $150^\circ$ ). Fish-eye and wide angle lenses offer a wider field of view, but require suitable projection models. Popular choices are the Brown-Conrady (BC) model [283], the Kannala-Brandt (KB) model [536] and the Double Sphere (DS) model [1114]. The 6-parameter DS model provides a comparable reprojection accuracy as the 8-parameter KB model while offering around five times faster computation time for the projection function. (©2018 IEEE)

### 7.3.1.1 Geometric Camera Models

**Perspective Cameras.** The projection function is generically referred to as: *projection*

$$\mathbf{z} = \pi(\mathbf{x}^c, \boldsymbol{\xi}),$$

where  $\mathbf{x}^c = [x \ y \ z]^\top \in \mathbb{R}^3$  is a 3D point in camera coordinates,  $\mathbf{z} = [u \ v]^\top \in \Omega \subset \mathbb{R}^2$  are the coordinates of the corresponding projected 2D point in the image domain  $\Omega$ , and  $\boldsymbol{\xi} \in \mathbb{R}^n$  represents the intrinsic parameters of the camera, typically precalibrated in visual SLAM. The dimensionality of  $\boldsymbol{\xi}$  depends on the used *camera model*.

Conversely, the unprojection operation is denoted as:

$$\mathbf{x}^c = \pi^{-1}(\mathbf{z}, \boldsymbol{\xi}),$$

which reconstructs a ray in 3D from a 2D image point  $\mathbf{z}$ . Since the depth of the point remains unknown, the resulting  $\mathbf{x}^c$  is only known up to scale.

*pinhole camera model*

In practice, there exists a wide range of projection functions suitable for different lens geometries and camera types (see some, for example, in [684]). Rectilinear models (also known as pinhole- or perspective camera model) are the simplest and can be used for narrow-angle lenses without distortion

$$\boldsymbol{\xi}_p = [f_u \ f_v \ u_0 \ v_0]^\top, \ \pi_p(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u \frac{x}{z} + u_0 \\ f_v \frac{y}{z} + v_0 \end{bmatrix}, \ \pi_p^{-1}(\mathbf{z}, \boldsymbol{\xi}) \sim \begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \\ 1 \end{bmatrix},$$

where  $f_u$  and  $f_v$  stand for the focal length in horizontal and vertical direction, in pixel units.  $u_0$  and  $v_0$  stand for the 2D coordinates of the principal point, and we assumed rectangular pixels. With typical, square pixels, we expect  $f_u \approx f_v$ .

*distortion*  
*radial-tangential*

To account for some amount of lens distortion, the radial-tangential model is

most commonly used:

$$\boldsymbol{\xi}_{RT} = [\boldsymbol{\xi}_p^\top \ k_1 \ k_2 \ p_1 \ p_2]^\top, \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix}, \quad r = \sqrt{x'^2 + y'^2},$$

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'(1 + k_1 r^2 + k_2 r^4) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y'(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{bmatrix}, \quad \pi_p(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u x'' + u_0 \\ f_v y'' + v_0 \end{bmatrix}.$$

Note that we cannot extract an analytical expression for the unprojection model  $\pi_{RT}^{-1}(\mathbf{z}, \boldsymbol{\xi})$ , since there is no analytical solution for  $x', y'$  as a function of  $x'', y''$  (*i.e.*, undistortion). We may, however, resort to iterative approaches, *e.g.*, the Newton-Raphson method.

**Wide-angle and Fisheye Cameras.** For wide-angle lenses —see Figure 7.1— *fisheye camera* up to fields of view (FOV) of  $180^\circ$ , pinhole models with a few radial distortion coefficients typically suffice. The Brown-Conrady model [283] amounts to the *Brown-Conrady* radial-tangential model described above without tangential coefficients, *i.e.*,  $\boldsymbol{\xi}_{BC} = [\boldsymbol{\xi}_p^\top \ k_1 \ k_2]$ . Note that despite the simplification relative to the radial-tangential model, no analytical undistortion exists.

For fish-eye lenses with FOVs larger than  $180^\circ$ , the Kannala-Brandt (KB) model *Kannala-Brandt* [536] has been used, for example in [142]:

$$\boldsymbol{\xi}_{KB} = [\boldsymbol{\xi}_p^\top \ \mathbf{k}_{KB}^\top]^\top, \quad \pi_{KB}(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u r(\theta) \cos \psi + u_0 \\ f_v r(\theta) \sin \psi + v_0 \end{bmatrix}^\top.$$

Here, the incoming ray is parametrized by the angles  $\theta = \arctan \frac{\sqrt{x^2+y^2}}{z}$  and  $\psi = \arctan \frac{y}{x}$ , distortion is parametrized by four coefficients  $\mathbf{k}_{KB} = [k_1 \ \dots \ k_4]^\top$  and the distortion expression is  $r(\theta) = \theta + \sum_1^4 k_n \theta^{2n+1}$ .

For fish-eye and wide angle lenses, a popular alternative is the Double Sphere (DS) model [1114] as it offers a good compromise of accuracy and speed. In the DS *double sphere* model a point is consecutively projected onto two unit spheres with centers shifted by  $\gamma$ . Then, the point is projected onto the image plane using the pinhole model shifted by  $\frac{\alpha}{1-\alpha}$ . This leads to:

$$\pi_{DS}(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u \frac{u}{\alpha d_2 + (1-\alpha)(\gamma d_1 + z)} + c_u \\ f_v \frac{v}{\alpha d_2 + (1-\alpha)(\gamma d_1 + z)} + c_v \end{bmatrix}^\top, \text{ with } \boldsymbol{\xi} = [\boldsymbol{\xi}_p^\top \ c_u \ c_v \ \gamma \ \alpha]^\top.$$

As shown in [1114], this model offers a closed-form unprojection solution. As a consequence, the 6-parameter DS model is around five times faster to compute than the 8-parameter KB model while offering a comparable reprojection error.

### 7.3.1.2 Photometric Models

The photometric calibration maps irradiance to pixel values:

*photometric calibration*

$$I = f(E, T),$$

where  $I$  is the pixel intensity,  $E$  is the irradiance, and  $T$  contains other camera properties that affect this mapping such as exposure time, analog and digital gain, gamma correction, de-bayering, and lens vignetting. This photometric calibration is typically only important up to scale and for methods that aim to obtain dense, textured scene representations or that rely on photo-consistency across images.

*rolling shutter*  
*global shutter*

#### 7.3.1.3 Time-Dependent Effects

In situations where the camera is moving while the image is taken —which is always almost the case for SLAM or Visual Odometry systems— it is important to also consider the effects of this motion on the image formation process. Most modern consumer cameras use a rolling shutter, which captures image-rows sequentially in time. In contrast, global shutter cameras, which are often used for machine perception applications, capture all image rows simultaneously.

In practice, it is advisable to either use global shutter cameras, or include the rolling shutter effect into the camera model by varying the camera pose for individual image rows. Note that this becomes significantly more practical in visual-inertial systems, where the IMU effectively measures local motion with a cameras exposure window, and thus simplifies the use and modeling of rolling shutter cameras significantly.

*keypoints*

*keypoint detector*

*keypoint descriptor*

#### 7.3.1.4 Practical Considerations

When selecting a camera model, the primary goal is to ensure that the parametric model can effectively and accurately approximate the behavior of the sensor and lens system. When using Fisheye lenses, using a spherical model is recommended —while for rectilinear lenses, linear base-models should be used. More generally, choosing a camera model involves balancing computational efficiency against precision: using an ill-suited camera model can introduce inaccuracies and systematic biases, significantly degrading the visual SLAM system’s accuracy and robustness.

### 7.3.2 Keypoints

The success of SLAM systems in mapping environments and providing accurate localization hinges on their ability to detect, describe, and match key features in a scene —commonly referred to as ‘*keypoints*’ or ‘*features*’. In visual SLAM, key-point detection involves identifying salient image regions that are distinctive and repeatable, ensuring reliable re-detection from different viewpoints, across multiple runs, and under varying conditions. An ideal **keypoint detector** should maintain these properties regardless of changes in illumination, viewpoint, or occlusions. Once detected, a **keypoint descriptor** encodes the local appearance of the corresponding keypoint detection into compact and distinctive representations.

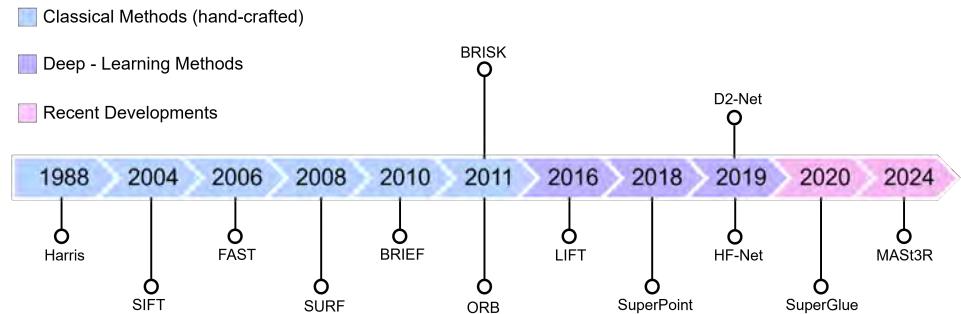


Figure 7.2 Timeline of some of the most prominent algorithms shaping the literature on visual keypoints for vision-based SLAM and image matching.

Ideally, a descriptor should uniquely characterize a keypoint's surroundings while remaining invariant to transformations such as lighting changes, rotation, or scale variations. This ensures both high recall — matching the same keypoint across different conditions — and high precision — avoiding incorrect correspondences with similar-looking but unrelated features.

In practice, real-world challenges such as drastic illumination variations, textureless surfaces, and dynamic scenes introduce errors in detection and matching, directly impacting SLAM performance. Consequently, research has focused on designing keypoints with specific characteristics to enhance robustness. The most desirable properties include **scale and rotation invariance** (ensuring consistent detection despite viewpoint changes), **repeatability and distinctiveness** (allowing reliable re-detection and unique identification), and **efficiency** (enabling real-time operation under computational constraints). To meet these demands, keypoint detection and description have evolved from classical handcrafted techniques to deep-learning-based approaches more recently. The following subsections provide an overview of the most prominent keypoint methods in the literature, as illustrated in Figure 7.2, along with emerging trends in the field.

### 7.3.2.1 Classical Keypoint Detectors & Descriptors

Classical hand-crafted techniques have played a pivotal role in the evolution of feature detection and description. Among them, the Harris-Stephens keypoint detector [432], widely known as the ‘Harris corner detector,’ is one of the most influential methods. It identifies corners by analyzing the eigenvalues of the second-moment matrix within image patches, classifying a patch as a keypoint when both eigenvalues are large, indicating strong intensity variations in two orthogonal directions. The Shi-Tomasi corner detector [1001] is built on the same principle but directly uses the smaller eigenvalue for corner selection. In contrast, Harris defines a ‘cornerness’ response function to approximate the process for efficiency. While robust and

computationally efficient, the Harris detector lacks scale invariance, a limitation that later methods, such as SIFT and SURF, sought to address.

A seminal milestone in keypoint detection was the introduction of the Scale Invariant Feature Transform (SIFT) [696], which set a new standard for precision and recall across challenging settings. SIFT is highly invariant to scale and rotation and partially invariant to illumination changes. It follows a structured three-stage process: (1) detecting keypoints as extrema in a scale-space Difference of Gaussians (DoG) pyramid and refining their localization through a 3D quadratic fit, (2) assigning orientation based on the dominant local image gradient, and (3) computing a 128-dimensional descriptor from a histogram of discretized gradient orientations. However, SIFT's exceptional robustness comes at a high computational cost, making it more computationally expensive for real-time applications.

To improve efficiency, Features from Accelerated Segment Test (FAST) [943] was developed as a high-speed corner detector. It assesses pixel intensities in a circular neighborhood around the candidate keypoint location and employs an early rejection strategy to minimize computations. FAST further enhances speed using a decision tree and non-maximum suppression. However, unlike SIFT, it lacks scale invariance, making it sensitive to significant transformations. Speeded-Up Robust Features (SURF) [63] later improved efficiency by approximating SIFT using integral images and box filters instead of Gaussian derivatives, achieving a better balance between speed and robustness.

The need for fast yet robust descriptors led to the development of binary-based methods. Binary Robust Independent Elementary Features (BRIEF) [141] introduced a compact descriptor using binary intensity comparisons, enabling fast descriptor matching using the Hamming distance. While BRIEF lacks scale and rotation invariance, it demonstrated that local gradients, fundamental to SIFT's robustness, could be effectively captured through simplified binary tests. Inspired by this success, ORB [950] was proposed, building upon FAST keypoint detection and BRIEF description, while adding scale and rotation invariance through an image pyramid and intensity centroid method. At the same time, BRISK [641] was proposed, employing FAST or Harris corners on a scale-space pyramid, and incorporating invariance to rotation changes within a binary descriptor by identifying a dominant keypoint direction similar to SIFT. While the added rotation- and scale-invariance of methods such as ORB and BRISK have a small impact on the distinctiveness of the output keypoints, they have proven effective in visual SLAM and real-time robotics applications. When computational cost, however, is not a requirement (*e.g.*, in Computer Vision applications), SIFT and SURF might still be preferable as they still offer greater robustness under challenging lighting or perspective changes.

With several variants of these methods appearing in the literature, classical hand-crafted keypoint detection and description methods have been foundational in Computer Vision and Robotics, carefully balancing robustness, efficiency, and

invariance. However, the demand for keypoints that adapt to increasingly complex, dynamic environments remains an ongoing challenge. This has driven research toward learning-based approaches, which aim to automatically optimize keypoints for real-world applications, setting the stage for the next generation of feature detection techniques.

#### 7.3.2.2 Deep-Learning-based Keypoint Detection & Description

By the late 2010s, deep learning-based approaches for image keypoints began to gain traction, utilizing large-scale datasets and Convolutional Neural Networks (CNNs) to learn robust features directly from data. Unlike manually designed keypoints, these methods automatically discover and optimize feature representations, demonstrating unparalleled adaptability and accuracy in scenarios that were previously infeasible. For instance, they have shown remarkable success in detecting stable keypoints even under extreme illumination changes—an area where classical hand-crafted methods struggled. Consequently, the visual SLAM community has increasingly shifted away from traditional feature engineering, embracing data-driven representation learning for keypoint detection and description.

The Learned Invariant Feature Transform (LIFT) [1242] was one of the first deep learning-based approaches to integrate keypoint detection, orientation estimation, and descriptor computation into an end-to-end trainable pipeline. Using CNNs to extract features from small image patches, LIFT achieved greater robustness to scale, illumination, and rotation changes than classical methods. It employs a sequential learning strategy, training descriptors first, followed by orientation estimation and then keypoint detection, ensuring stable and effective feature extraction. Similarly, SuperPoint [267] introduced a self-supervised framework that detects keypoints and computes descriptors in a single forward pass, making it efficient for real-time applications. Unlike patch-based networks, SuperPoint operates on entire images and leverages homographic adaptation, a self-supervised learning technique to generate pseudo-ground truth keypoints. This approach significantly improves keypoint repeatability and descriptor quality compared to classical methods, such as SIFT, ORB, and FAST, especially under illumination changes.

Building on these advancements, HF-Net [967] introduced a hierarchical localization approach that combines global image retrieval with precise local feature matching. HF-Net improves computational efficiency while maintaining high robustness by integrating keypoint detection, local descriptors, and global descriptors into a single CNN. This architecture reduces runtime by limiting the number of images used for matching, making it particularly effective for large-scale SLAM and real-time applications, even under extreme appearance variations such as night-time scenes. HF-Net's learned features are sparser but more discriminative than those of SuperPoint, rendering it a preferred choice for deep learning-based SLAM systems such as DX-SLAM [647].

Interestingly, D2-Net [295] introduced a describe-and-detect approach that re-

verses the traditional keypoint detection and descriptor extraction order. Instead of detecting keypoints first, D2-Net computes dense feature maps using a CNN and then identifies keypoints as local maxima within these maps. This method captures high-level semantic information, making it robust to extreme lighting changes and weakly textured environments. Unlike SuperPoint, which separates detection and description, D2-Net jointly optimizes both tasks, enhancing descriptor consistency. However, while this dense approach improves robustness, it is computationally more demanding than classical sparse methods. Despite this trade-off, D2-Net remains highly effective for visual localization and SfM tasks, pushing the boundaries of deep learning-based feature extraction.

### 7.3.3 Reprojection Error

*reprojection error*

The visual reprojection error measures the discrepancy between observed image points  $\mathbf{z}_j \in \mathbb{R}^2$  and reprojected points  $\pi(\mathbf{x}_j^c, \boldsymbol{\xi}) \in \mathbb{R}^2$ :

$$\mathbf{e}_{\text{reproj}} = \mathbf{z}_j - \pi(\mathbf{x}_j^c, \boldsymbol{\xi}),$$

where  $\mathbf{z}_j$  is the observed 2D image point,  $\mathbf{x}_j^c \in \mathbb{R}^3$  is the 3D point in camera coordinates,  $\boldsymbol{\xi}$  are the camera intrinsic calibration parameters, and  $\pi$  is the projection function.

Assuming the observed image points  $\mathbf{z}_j$  are perturbed by Gaussian noise, the likelihood function can be expressed as:

$$p(\mathbf{z}_j | \mathbf{x}_j^c, \boldsymbol{\xi}) \sim \mathcal{N}(\pi(\mathbf{x}_j^c, \boldsymbol{\xi}), \boldsymbol{\Sigma}_j),$$

where  $\boldsymbol{\Sigma}_j$  is the covariance of the Gaussian noise of the position of the feature in the image. In the simplest case, the noise is assumed to be isotropic and constant along the image  $\boldsymbol{\Sigma}_j = \sigma \mathbf{I}_2$ .

Maximizing the likelihood of a set of observations is equivalent to minimizing their negative log-likelihood:

$$\mathcal{L} = - \sum_j \log p(\mathbf{z}_j | \mathbf{x}_j^c, \boldsymbol{\xi}) = \frac{1}{2} \sum_j \|\mathbf{z}_j - \pi(\mathbf{x}_j^c, \boldsymbol{\xi})\|_{\boldsymbol{\Sigma}_j}^2 + \text{const.}$$

Removing the constant, we get the weighted-squared reprojection error of the set of points observed in an image:

$$E_{\text{reproj}} = \frac{1}{2} \sum_j \|\mathbf{z}_j - \pi(\mathbf{x}_j^c, \boldsymbol{\xi})\|_{\boldsymbol{\Sigma}_j}^2. \quad (7.1)$$

*robust kernels*

To handle outliers, robust kernels such as the Huber or Tukey kernel are applied (*cf.* Chapter 3). For example, the robust reprojection error can be expressed as:

$$E_{\text{robust}} = \frac{1}{2} \sum_j \rho \left( \|\mathbf{z}_j - \pi(\mathbf{x}_j^c, \boldsymbol{\xi})\|_{\boldsymbol{\Sigma}_j} \right), \quad (7.2)$$

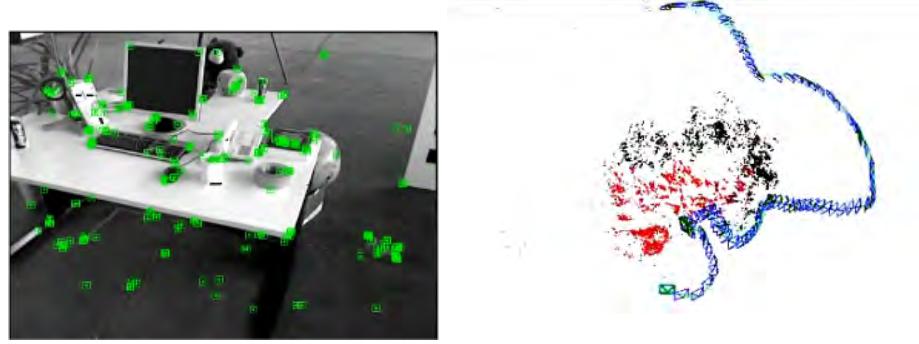


Figure 7.3 The feature-based visual SLAM problem: given a set of features matched in each image (left), estimate the position of their corresponding 3D points and the pose from where each image was acquired (right). Images generated with ORB-SLAM [784].

where  $\rho(\cdot)$  is a robust kernel function that reduces the influence of large residuals. In the following, for simplicity, we drop the dependence on the camera intrinsics  $\xi$ .

#### 7.3.4 Keypoint-Based Visual SLAM

The core of feature-based visual SLAM is minimizing the reprojection error. Suppose we have a set of environment points  $P_j \in \mathbb{P}$  that are observed in a set of images  $C_i \in \mathcal{C}$  taken with a moving camera. To make notation simpler, we will just use the point and camera identifiers writing  $j \in \mathbb{P}$  and  $i \in \mathcal{C}$ . The goal of visual SLAM is to estimate the point coordinates  $\mathbf{x}_j^w \in \mathbb{R}^3$  in the world reference frame  $\mathcal{F}^w$ , as well as the poses of the camera  $\mathbf{T}_w^i \in \text{SE}(3)$ . In the monocular case, the observation of each point in each image is just the observed image coordinates  $\mathbf{z}_{ij} \in \Omega_i \subset \mathbb{R}^2$  (Figure 7.3).

The minimization of the reprojection error, when applied to all the points and camera poses, is known as *full BA*:

$$\{\mathbf{T}_w^i, \mathbf{x}_j^w | i \in \mathcal{C}, j \in \mathbb{P}\} = \arg \min_{\mathbf{T}_w^i, \mathbf{x}_j^w} \frac{1}{2} \sum_{i,j} \rho \left( \|\mathbf{z}_{ij} - \pi(\mathbf{R}_w^i \mathbf{x}_j^w + \mathbf{t}_w^i)\|_{\Sigma_{ij}} \right). \quad (7.3)$$

Typically, researchers tackle the reprojection error optimization via several advanced methods, each offering unique trade-offs in computational complexity and robustness:

- **Batch Optimization:** The overall reprojection error is iteratively minimized over all observations w.r.t. all poses and landmarks, *i.e.*, solving Equation (7.3). Popular algorithms for minimization are the Gauss-Newton (GN) and Levenberg-Marquardt (LM). This method is the gold standard in SfM, but is too expensive to run for each image in real-time SLAM. batch optimization

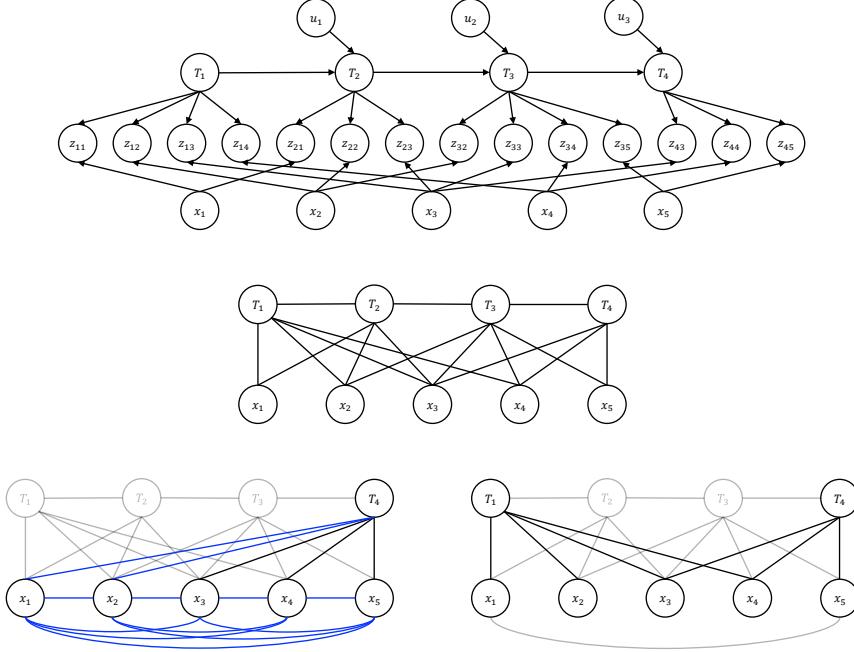


Figure 7.4 A visual SLAM example with four camera poses and five features: Bayes network (top), and its corresponding Markov random field (middle). EKF SLAM marginalizes out past camera poses, resulting in a dense graph (lower left). Keyframe SLAM keeps just a few camera poses and discards observations from intermediate images, keeping SLAM sparsity (lower right) [1036].

*filtering*

- **Filtering-Based Approaches:** Use sequential and causal methods like the Extended Kalman Filter (EKF) or Information Filter for real-time state estimation. They simplify the problem by keeping only the last camera pose, but unfortunately this destroys sparsity (Figure 7.4), limiting them to a few hundred features. Also, filtering methods do not re-linearize past observations, losing accuracy.

*keyframe*

- **Keyframe-Based Approaches:** They simplify the problem by keeping just a few images, called keyframes [582]. Intermediate images and their observations are discarded for the map estimation. For the same computational effort, they can build longer and more accurate maps than filtering methods [1036].

*factor graphs*

- **Factor Graphs:** All the above approaches can be formalized by means of factor graphs. To this end, one represents SLAM problems as graphs where nodes correspond to variables (*e.g.*, poses, landmarks) and factors represent the reprojection errors and priors on calibration and/or camera poses, which is discussed in detail in Part I of this book.

### 7.3.5 Photometric Error and Direct Methods

The photometric error provides an alternative to the reprojection error by offering to directly minimize the differences in pixel intensities between observed and projected image regions. This approach is rooted in the principle of photometric consistency, which assumes that corresponding pixels in consecutive frames represent the same scene point under consistent lighting conditions.

*direct methods*

### 7.3.6 Visual Place Recognition and Global Localization

Visual Place Recognition consists of, given a query image, finding one from the same place in a database of registered images. This is typically solved by computing a per-image descriptor  $\mathbf{d} = f(I) \in \mathbb{R}^d$  that summarizes the content of the image, and retrieving the closest one in this descriptor space via k-NN search. Galvez-Lopez and Tardos [362] introduced bag-of-words approaches that basically aggregate ORB or other descriptors by their quantization into visual clusters or “words”. While they excel at small temporal and spatial ranges, they are limited by the low invariance of hand-crafted features to variations of visual textures. For these cases, deep architectures have been proposed and trained for feature extraction and aggregation with high invariance to visual appearance changes [35, 508].

*visual place recognition*

### 7.3.7 Initialization

The minimization of the reprojection error of Equation 7.3 is typically addressed by non-linear iterative optimization, which requires sufficiently accurate initial guesses to converge. The initialization of the visual SLAM states in the first frames of a video sequence is hence relevant for a correct tracking, in particular for monocular setups, where the full state is not observable from a single frame.

### 7.3.8 Map Representations

Map representations are a critical aspect of visual SLAM systems, as they define how the environment is modeled and stored for navigation, mapping, and localization purposes. A well-designed map representation strikes a balance between accuracy, memory efficiency, and computational cost. We refer the reader to Chapter 5 for a broader discussion about dense map representations.

## 7.4 Further Considerations about Image Alignment and BA

Here we provide more details about image alignment (*i.e.*, how to estimate the current camera pose from images—which is key to both odometry and loop closure detection) and briefly mention solution techniques for BA.

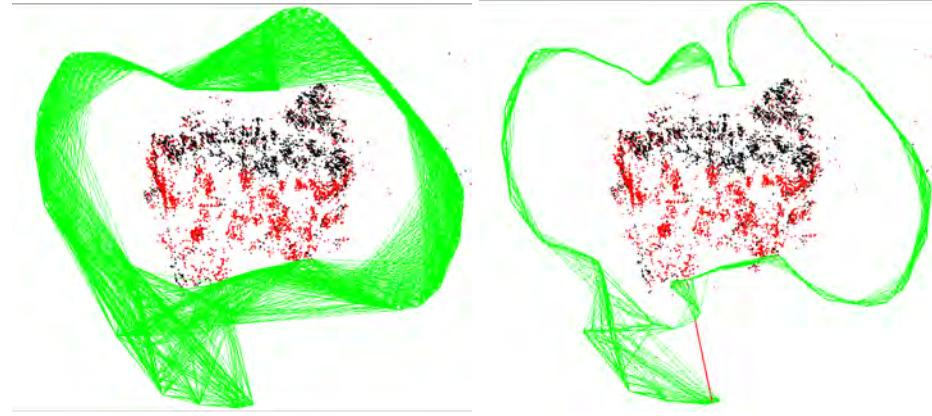


Figure 7.5 Representation of locality in ORB-SLAM [784]. The covisibility graph (left) connects keyframes that have seen at least  $\theta$  points in common (in this example  $\theta = 15$ ) and is used for local BA. The essential graph (right) is a sparser version, that in this example connects keyframes with at least  $\theta = 100$  points in common, and is used for pose-graph optimization during loop correction. ©2015 IEEE.

#### 7.4.1 Keypoint-based Image Alignment

Although full BA (equation 7.3) is the gold standard in SfM, it is too expensive to run at frame rate, which is typically between 10 and 50 Hz. To operate in real-time, most keypoint-based visual SLAM pipelines use two key ideas:

**Parallel Tracking and Mapping.** splitting the SLAM process into two threads that run in parallel [582]:

- A tracking thread that finds feature matchings for the current image  $i \in C$  and computes its camera pose, without updating the estimated map points, using *pose-only BA*:

$$\mathbf{T}_w^i = \arg \min_{\mathbf{T}_w^i} \sum_{j \in P} \rho \left( \| \mathbf{z}_{ij} - \pi(\mathbf{R}_w^i \mathbf{x}_j^w + \mathbf{t}_w^i) \|_{\Sigma_{ij}} \right). \quad (7.4)$$

- A mapping thread that solves BA only for a subset of images  $K \subset C$  called *keyframes*, whose poses are the only ones that will be included in the map. In this way, BA only needs to run at keyframe rate, typically between 0.5 and 5 Hz. Keyframes can be inserted at a constant frequency, but a more sensible option is to upgrade to keyframes those frames that contain significantly new information.

*pose-only BA*

*local BA*

**Locality.** when the camera is operating in a large environment, its observations have a negligible effect on the parts of the map that are far away, except in loop closure events. The usual approach is to relegate loop correction to a third thread that runs quite infrequently, and to run *local BA* in a window of keyframes in the

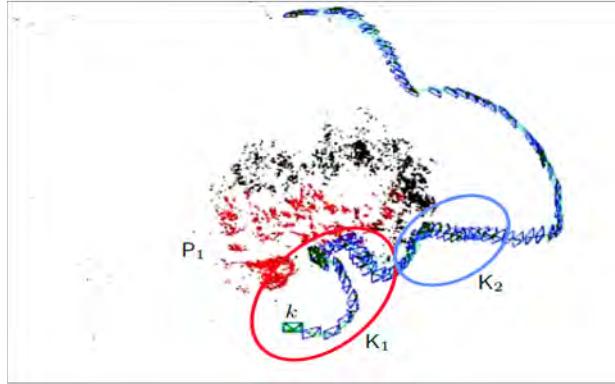


Figure 7.6 Implementation of local BA in ORB-SLAM [784]. The local map is defined by the set of keyframes  $K_1$  that contains the current keyframe  $k$  and its neighbors in the covisibility graph, and the set  $P_1$  of points seen by them (in red).  $K_2$  is the rest of keyframes in the map that see some point from  $P_1$ .

mapping thread. The local window can be defined using a temporal criterion as the last  $k$  frames or keyframes, which is the usual choice in visual odometry or visual-inertial SLAM systems. In visual SLAM systems, a better option is to base the local window on a *covisibility* criterion, for example, including in the local window keyframes that have more than  $\theta$  observed points in common with the current keyframe [1035, 784] (see example in Figure 7.5). In that way, *local BA* can update just a set of covisible keyframes and the points observed by them (Figure 7.6):

$$\{\mathbf{T}_w^{k^*}, \mathbf{x}_j^{w^*} \mid k \in K_1, j \in P_1\} = \arg \min_{\mathbf{T}_w^k, \mathbf{x}_j^w} \frac{1}{2} \sum_{\substack{i \in K_1 \cup K_2, \\ j \in P_1}} \rho \left( \|z_{ij} - \pi(\mathbf{R}_w^i \mathbf{x}_j^w + \mathbf{t}_w^i)\|_{\Sigma_{ij}} \right). \quad (7.5)$$

An example of a complete visual SLAM pipeline can be seen in Figure 7.7 with four threads: tracking that runs at frame rate, local mapping that runs at keyframe rate, loop closing that tries to detect loops for every keyframe and corrects them when detected, and full BA that can be run optionally to improve the map after a loop closure.

#### 7.4.2 Direct Image Alignment

Direct methods such as LSD-SLAM [307] or DSO [308] typically pursue a similar pipeline of first estimating a frame-to-frame tracking and mapping and then assuring some form of global consistency. Rather than first extracting, matching and tracking points and subsequently minimizing a geometric reprojection error, however, they directly use the brightness information from the sensor and aim to compute the maximum a posteriori estimate of 3D structure and camera motion

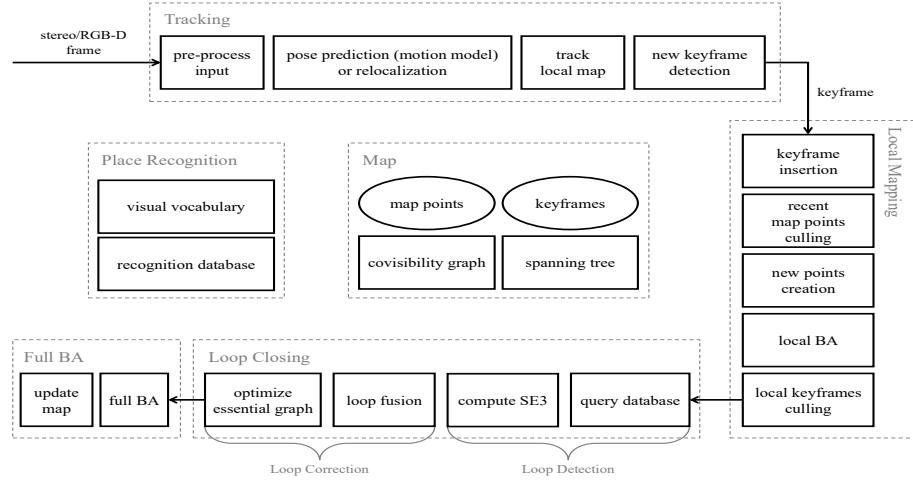


Figure 7.7 Structure of ORB-SLAM2 system [782] showing the map, the place recognition database and its complete processing pipeline composed of four threads: tracking, local mapping, loop closing and full BA. ©2017 IEEE.

given the raw sensory data. This amounts to solving the correspondence estimation and SLAM problem jointly by minimizing a photometric loss of the form:

$$E_{photo} = \sum_{i \in \mathcal{F}} \sum_{z \in \mathcal{P}_i} \sum_{j \in obs(z)} \rho(I_i(z) - I_j(\omega(z, d_z, \mathbf{T}_j^i)), \quad (7.6)$$

with respect to all camera parameters  $\mathbf{T}_j^i \in SO(3)$  and all depth values  $d_z \in \mathbb{R}$ . This loss assures that the colors  $I_i$  and  $I_j$  of corresponding points in all frame pairs  $i$  and  $j$  are consistent. More specifically, we sum over the set of all keyframes  $\mathcal{F}$  and for every point  $z$  in keyframe  $\mathcal{P}_i$ , we assure color consistency for all the frames  $obs(z)$  where this point is visible. The warping  $w$  takes the point  $z$  with its depth value  $d_z$ , transforms it from frame  $i$  to frame  $j$  with the rigid body motion  $\mathbf{T}_j^i$  and perspectively projects it back into image  $I_j$ .

As shown in Figure 7.8, LSD-SLAM optimizes for depth maps and camera motion (tracking) in alternation and performs a pose-graph optimization (PGO) to assure global consistency of the estimated camera motion with the computed pairwise image alignments.

In contrast, DSO [308] jointly optimizes for structure and motion in the form of a photometric BA. The robust loss  $\rho$  is implemented by weighted sum of squared differences over a small patch that includes an automatic exposure time adaptation (in case the exposure time is unknown). The dependency of respective residuals is captured in form of factor graphs. And real-time performance on a CPU is achieved by limiting the optimization to a sliding window of a subset of keyframes while marginalizing out the effect of older frames. This leads to a significant boost in

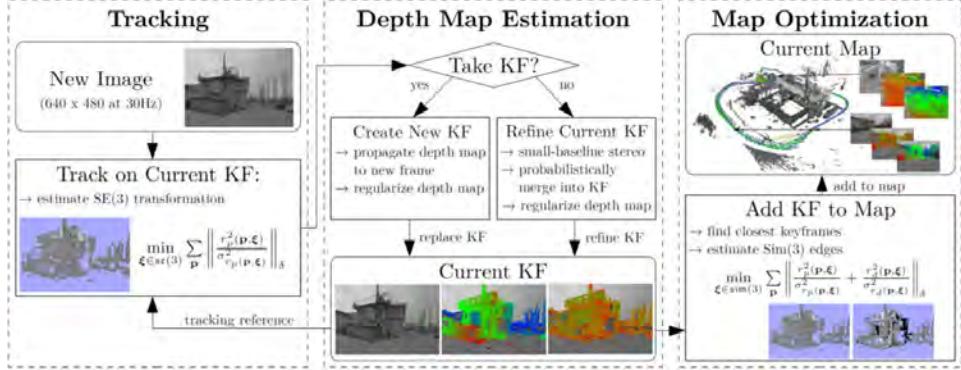


Figure 7.8 Schematic overview of Large Scale Direct (LSD) SLAM [307] showing the three components that perform direct camera tracking, direct mapping and PGO for global consistency, all running in alternation. In contrast, Direct Sparse Odometry [308] performs the optimization of structure and motion in a single Gauss-Newton optimization in order to achieve even higher precision. Direct methods like DSO were shown to provide higher precision than keypoint-based methods because they do not perform any intermediate abstraction and can determine camera motion from even very subtle brightness variations [308] (©2017 Springer).

precision since it relies on a statistically optimal estimate of structure and motion given all the sensory brightness data.

In real-time-capable SLAM methods, global consistency can be achieved in several steps: Firstly, one can jointly optimize over the last  $k$  keyframes as done in DSO to assure consistency over a sliding temporal window (sliding window photometric BA). Secondly, one can additionally run a PGO [552, 365] —see Figure 7.9— in order to recompute a camera trajectory that is maximally consistent with all estimated pairwise image alignments. And thirdly, one can perform an adaptive version of PGO called Pose Graph Bundle Adjustment (PGBA) [1132] which additionally incorporates the full photometric uncertainty of BA with the same computational efficiency (because only the camera poses are being updated).

#### 7.4.3 Solving BA

Although BA could be solved using general variable elimination techniques as discussed in Chapter 2, there are specialized solvers that able to benefit from the sparsity structure of the problem. Figure 7.10 shows a toy example with 4 cameras and 9 points observed from them. The observation Jacobian is very sparse, as each observation  $\mathbf{z}_{ij}$  depends only on the camera  $i$  and the point  $j$ . As a result, each observation introduces in the Hessian a diagonal block for the camera, a diagonal block for the point, and an off-diagonal camera-point block. As the number of points

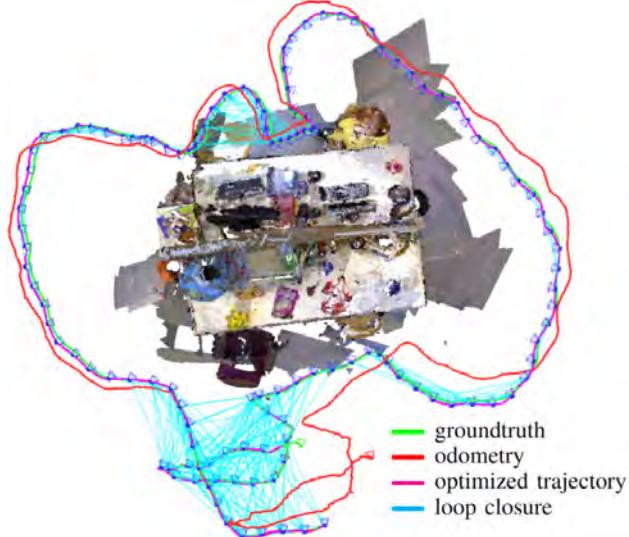


Figure 7.9 In direct visual SLAM methods, one can perform pose-graph optimization in order to compute a trajectory that is globally consistent with all previously estimated pairwise image alignments [552]. (©2013 IEEE)

is typically several orders of magnitude larger than the number of cameras, a good solution is to eliminate first the points, and then solve for the cameras.

This can be done using the Schur complement. If we have a linear system where  $\mathbf{D}$  is invertible, we can transform it by multiplying both sides on the left by a matrix:

$$\begin{aligned} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} &= \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \\ \begin{pmatrix} \mathbf{I} & -\mathbf{BD}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} &= \begin{pmatrix} \mathbf{I} & -\mathbf{BD}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \\ \begin{pmatrix} \mathbf{A} - \mathbf{BD}^{-1}\mathbf{C} & \mathbf{0} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} &= \begin{pmatrix} \mathbf{b}_1 - \mathbf{BD}^{-1}\mathbf{b}_2 \\ \mathbf{b}_2 \end{pmatrix}, \end{aligned}$$

to get a system where we can solve first  $\mathbf{x}_1$  and then  $\mathbf{x}_2$ :

$$\begin{aligned} (\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C}) \mathbf{x}_1 &= \mathbf{b}_1 - \mathbf{BD}^{-1}\mathbf{b}_2, \\ \mathbf{D} \mathbf{x}_2 &= \mathbf{b}_2 - \mathbf{C}\mathbf{x}_1. \end{aligned}$$

The BA problem needs to solve in each iteration an equation of the form:

$$\begin{pmatrix} \mathbf{H}_{cc} & \mathbf{H}_{cp} \\ \mathbf{H}_{cp}^\top & \mathbf{H}_{pp} \end{pmatrix} \begin{pmatrix} \mathbf{d}_c \\ \mathbf{d}_p \end{pmatrix} = \begin{pmatrix} \mathbf{b}_c \\ \mathbf{b}_p \end{pmatrix}.$$

This can be solved in three steps: computing the Schur complement of the points to obtain the reduced camera system, solving it for the cameras, and finally solving for the points:

$$\begin{aligned} \mathbf{H}_{cc}^{red} &= \mathbf{H}_{cc} - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top, \\ \mathbf{H}_{cc}^{red} \mathbf{d}_c &= \mathbf{b}_c - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{b}_p, \\ \mathbf{H}_{pp} \mathbf{d}_p &= \mathbf{b}_p - \mathbf{H}_{cp}^\top \mathbf{d}_c. \end{aligned} \quad (7.7)$$

As  $\mathbf{H}_{pp}$  is block diagonal, the Schur complement and the final point solution can be done very efficiently point by point. As shown in Figure 7.10 the reduced camera system is less sparse, as it contains blocks that relate pairs of cameras that have seen some point in common. In the example, there is a block between cameras 1 and 2, but not between cameras 1 and 3. In local BA the reduced camera system will be almost full and can use dense matrix solvers. In contrast, full BA has a much larger number of keyframes but there is less covisibility between them, so it can profit from a sparse solver for the reduced camera system.

#### 7.4.4 Bundle Adjustment Revisited

BA has been studied for over a century and the classical approach detailed in Section 7.4.3 has been established and shown to work well in a multitude of seminal papers [1104, 14, 975]. Yet, this pipeline has two important shortcomings: Firstly, respective solutions require a suitable initialization of landmarks and camera poses. And secondly, for large-scale problems the computational and memory requirements can grow prohibitively large. In recent years, there have been a series of papers that address these shortcomings and challenge the traditional computational pipeline [469, 470, 258, 259, 1167, 1168, 1169].

The key computational bottleneck is the solution to the reduced camera system (7.7). Instead of solving this by means of an iterative conjugate gradient algorithm, Power Bundle Adjustment [1168] approximates the inverse of the Schur matrix

$$\mathbf{H}_{cc}^{red} = \mathbf{H}_{cc} - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top, \quad (7.8)$$

by means of a matrix power series [1168]:

$$(\mathbf{H}_{cc}^{red})^{-1} \approx \sum_{i=0}^m (\mathbf{H}_{cc}^{-1}\mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top)^i \mathbf{H}_{cc}^{-1}. \quad (7.9)$$

This power series provably converges to the true inverse for increasing cut off parameter  $m$  [1168]. The main advantage is that tedious matrix inversion is replaced

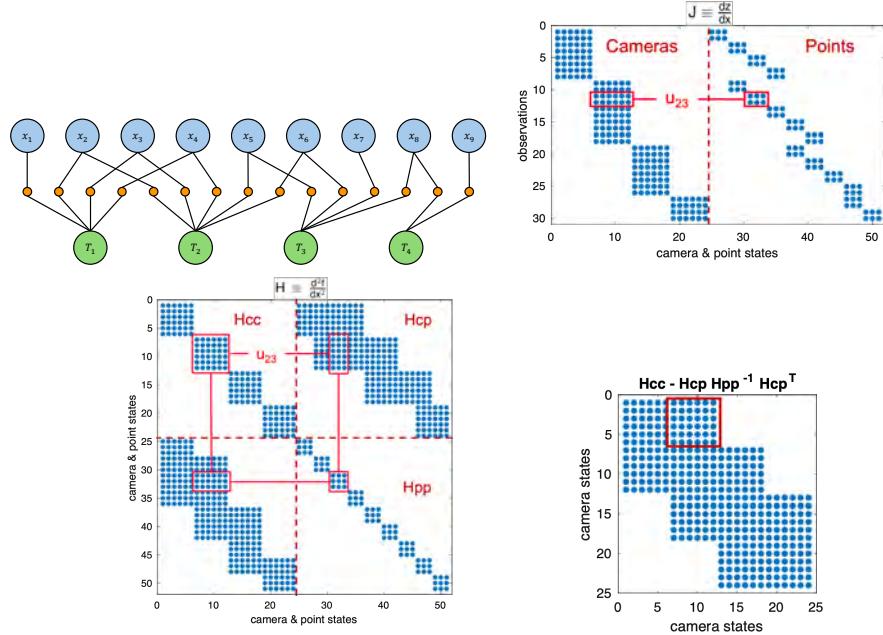


Figure 7.10 Toy example with 4 cameras and 9 points showing in the first row the factor graph and the Jacobian of point observations. The second row shows complete Hessian, and the Hessian of the reduced camera system.

by simple matrix multiplications, which can be done much faster and more memory-efficiently.

The dependency on initialization is alleviated in [469, 470] by reverting to the concept of variable projection—see Figure 7.11. To this end, the BA problem is split into two stages. In the first stage, the complicated perspective projection is replaced by a generic projective matrix such that the resulting optimization can be solved analytically for the landmarks as a function of the camera parameters. This removes the chicken-and-egg dependency between landmarks and camera poses, leading to a larger basin of attraction when optimizing the camera poses. In the second stage of projective refinement, one uses the computed solution as an initialization for the original (perspective) reconstruction. Although this strategy is computationally too demanding for more than 100 cameras, its combination with the power series approach as proposed in [1169] offers a scalable solution for large-scale BA problems without initialization.

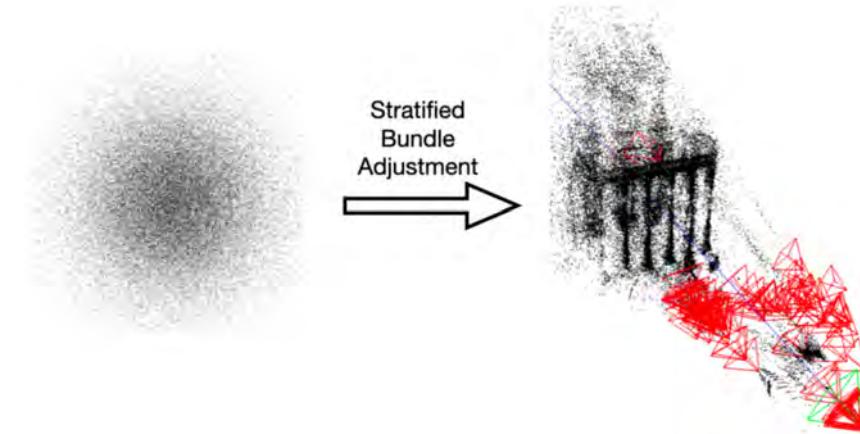


Figure 7.11 In a series of papers [469, 470, 1168, 1169], researchers advocate the use of variable projection methods and matrix power series in order to solve large scale bundle adjustment problems without initialization in a runtime- and memory efficient way. As shown above, camera poses and landmarks can thus be computed starting from a random initialization. (©2024 Springer)

## 7.5 Examples of Full Visual SLAM Systems

LSD-SLAM (Large-Scale Direct SLAM) [307] is a direct SLAM system that focuses on dense tracking and semi-dense mapping. It relies on photometric error minimization rather than keypoint-based methods, making it particularly effective in low-texture environments. The system operates in real-time, providing a semi-dense reconstruction of the scene and is well-suited for monocular cameras in indoor and small-scale outdoor environments.

ORB-SLAM [784, 782] is one of the most widely adopted SLAM systems due to its robustness and flexibility. It integrates keypoint-based tracking using ORB (Oriented FAST and Rotated BRIEF) descriptors, effective loop closure detection, and sparse map representation. ORB-SLAM supports monocular, stereo, and RGB-D cameras, making it highly versatile across different setups. It excels in scenarios requiring high accuracy and robust relocalization capabilities. In ORB-SLAM3 [142] it was extended to fisheye cameras, multimap, and visual-inertial SLAM. While originally conceived as a visual-inertial odometry system, OKVIS [643], the newest version, OKVIS2 [642], a visual-inertial SLAM system, may also be run in vision only (multi-camera) mode. Similar to the various ORB-SLAM versions, it uses keypoints and descriptors (BRISK).

Direct Sparse Odometry (DSO) [308] is a direct method for estimating 3D point cloud and camera trajectory. In contrast to LSD-SLAM camera motion and land-

mark points are estimated jointly in a single Gauss-Newton optimization. In order to achieve real-time performance, only the last  $k$  key frames are being updated resulting in a sliding-window photometric BA. The number  $k$  of considered keyframes provides a trade-off between speed and accuracy. Moreover, DSO makes use of a full photometric calibration with camera response function and vignette. The traditional brightness-constancy assumption is thus replaced by an irradiance-constancy assumption, *i.e.*, the assumption that respective points in the 3D world emit the same irradiance over time. Extensions of this approach to stereo systems [1157] and omni-directional cameras [736] have been proposed. Loop closure detection has also been added to reduce drift in longer sequences [365].

While among real-time capable approaches direct methods like DSO were shown to provide more accuracy and robustness than keypoint-based methods [308], for optimal performance they typically require a good photometric calibration and a global shutter camera. The rolling shutter effect leads to geometric distortions. While these can be modeled in direct SLAM methods [977, 978], the resulting approaches are often no longer real-time capable. As a result they are better handled in keypoint-based approaches which by design minimize geometric distortions. Nevertheless, direct methods often do better on low-resolution videos where due to blurring and down-sampling it may be harder to identify reliable feature points [1224]. Furthermore, feature points are valuable for efficient re-localization and loop closing. As a consequence, practical visual SLAM systems will often revert to hybrid approaches that combine the best of both worlds. An example of a hybrid approach is [385] where feature-based re-localization information is tightly integrated in a direct visual SLAM approach to further boost its robustness and precision.

## 7.6 Real-time Dense Reconstruction

While traditionally BA and SLAM tackle the problem of reconstructing camera motion and a sparse set of landmark points, for numerous applications ranging from augmented reality to autonomous robots one would prefer having a dense reconstruction of the observed world. To this end, a number of algorithms for real-time dense reconstruction from a monocular camera have been advocated over the years [1043, 797, 1177, 872]. Traditionally they revert to variational methods to estimate a continuous 3D structure by minimizing a loss function [1043]:

$$\min_{h:\Omega \rightarrow \mathbb{R}} \frac{1}{2} \sum_{i \in \mathcal{I}(x)} \int_{\Omega} \rho_i(\mathbf{z}, h) d\mathbf{z} + \lambda \int |\nabla h| d^2\mathbf{z}, \quad (7.10)$$

with respect to a dense map  $h$  that assigns a depth value to every pixel  $\mathbf{z}$  in the image plane  $\Omega \subset \mathbb{R}^2$ . The residual

$$\rho_i(\mathbf{z}, h) = \left| I_i \left( \pi \left( \mathbf{R}_w^i \mathbf{x}^w(\mathbf{z}, h) + \mathbf{t}_w^i \right) \right) - I_0(\mathbf{z}) \right|, \quad (7.11)$$

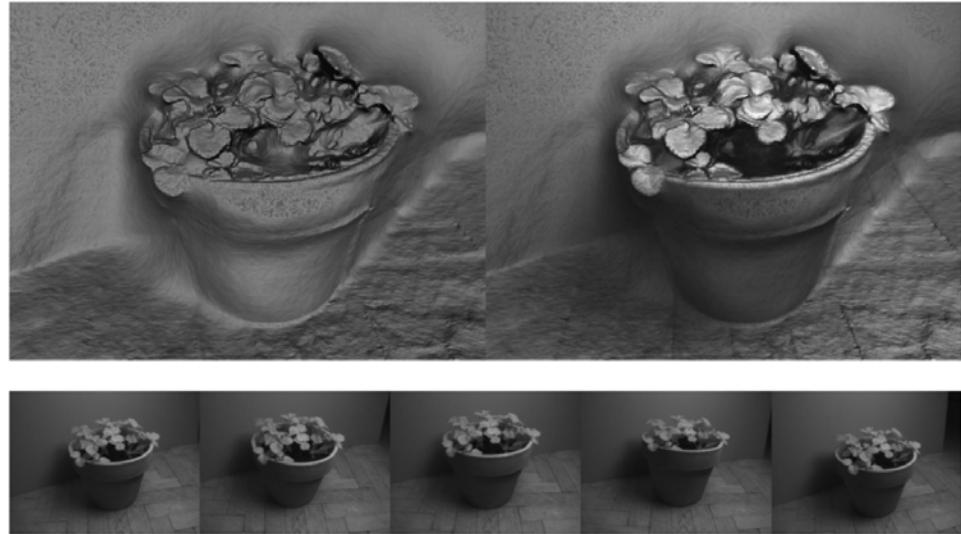


Figure 7.12 Dense reconstructions of the 3D world (above) can be computed in real-time from a handheld monocular camera (below) using variational methods that are efficiently parallelized on a GPU [1043]. Related approaches were proposed in [797, 1177, 872]. (©2010 Springer)

enforces consistency of the brightness at pixel  $\mathbf{z}$  in the reference image  $I_0$  to the corresponding pixels in a set of adjacent images  $I_i$ . The corresponding pixel is obtained by taking the 3D point  $\mathbf{x}^w(\mathbf{z}, h)$ , transforming it from the world frame by the corresponding rotation matrix  $\mathbf{R}_w^i \in SO(3)$  and translation vector  $\mathbf{t}_w^i \in \mathbb{R}^3$  to camera  $i$ , and finally projecting it by the model  $\pi(\cdot)$  into the image  $I_i$ .

The total variation regularizer (weighted by  $\lambda$ ) enforces spatial smoothness of the computed depth map and induces a soap-film-like fill-in for unobserved areas as shown in Figure 7.12.

## 7.7 SLAM with Depth-sensing Cameras

With the introduction of Microsoft Kinect, depth-sensing cameras became a commodity. These so-called RGB-D cameras are typically either structured-light or time-of-flight-based and provide a stream of depth images, often in conjunction with a color image stream. As such, they are conceptually between standard cameras and LiDAR sensors. Yet, in contrast to LiDAR sensors they provide an instant 2D array of depth values. Equipped with suitable algorithms, RGB-D cameras are very powerful for 3D sensing, albeit being limited to indoor applications (because the infrared-based sensing clashes with sunlight) and a certain range (typically up to around 5 meters).

*LiDAR*

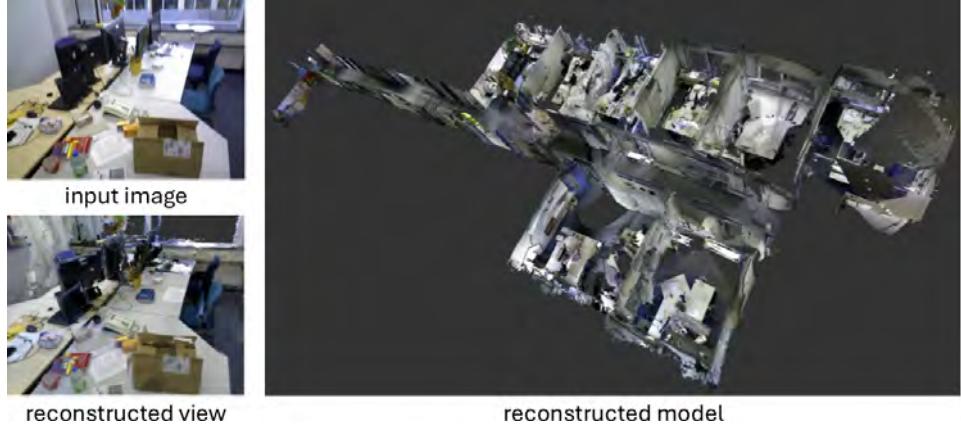


Figure 7.13 Dense reconstructions of a large-scale corridor scene with multiple offices computed from a moving RGB-D camera [1027]. Using octrees [1027] or voxel hashing [805] and direct camera tracking [553], such reconstructions were demonstrated to run in real-time on a GPU [1027] and even on a tablet CPU [1028] (©IEEE)

Kinect Fusion [798] built upon previous work for range image fusion [234] to advocate a method for reconstructing camera motion and 3D structure from a moving RGB-D camera. The basic idea for fusing the various depth images into a coherent 3D reconstruction is to encode each depth image as a (projective) signed distance function  $d_i(\mathbf{x})$  which for every voxel  $\mathbf{x} \in V$  encodes the (signed) distance to the nearest surface point (along the viewing ray). Subsequently one can compute an aggregated distance function  $D(\mathbf{x})$  for all voxels as a weighted average of the individual distance functions:

$$D(\mathbf{x}) = \frac{\sum_i \omega_i(\mathbf{x}) d_i(\mathbf{x})}{\sum_i \omega_i(\mathbf{x})}. \quad (7.12)$$

Under the assumption of Gaussian noise in the depth direction, this weighted average is nothing but the maximum-likelihood estimate of the distance function. For more robustness, one typically averages *truncated* signed distance functions so that each sensed surface point merely has a local impact on the reconstruction. The weights  $\omega_i(\mathbf{x})$  encode the certainty of the respective surface measurements (that is sensor dependent and typically decays with distance from the object). In order to fill holes in the reconstruction, one can revert to post-processing techniques [798], or modify the above weighting scheme to ensure watertight-ness of the reconstructions [1041].

While earlier RGB-D SLAM approaches track the camera by means of aligning respective depth point clouds with the ICP algorithm [78], subsequent works advocated a direct minimization of residuals that measure color and depth consistency

of two consecutive RGB-D frames  $(I_1, d_1)$  and  $(I_2, d_2)$  [553]:

$$r_I(\xi) = I_2(\tau_g(\mathbf{z})) - I_1(\mathbf{z}), \quad r_d = d_2(\tau_g(\mathbf{z})) - [g\pi^{-1}(\mathbf{z}, d_1(\mathbf{z}))]_z, \quad (7.13)$$

where  $g \in \text{SE}(3)$  denotes the desired rigid body motion,  $\tau_g(\mathbf{z}) = \pi g \pi^{-1}(x, d_1(\mathbf{z}))$  denotes the induced warping between corresponding pixels, and  $[\cdot]_z$  returns the  $z$ -component of a point. One can then fit the distribution of all residuals  $r = (r_I, r_d)$  with a suitable distribution and determine a maximum a posteriori estimate of the camera transformation  $g = \exp(\xi)$  by means of a coarse-to-fine Gauss-Newton optimization in  $\xi \in \text{se}(3)$  [1026, 552]. Compared to the classical ICP approach, this direct tracking approach reduces the root mean square tracking error by nearly an order of magnitude on established benchmarks while being significantly faster —see [553] for details.

For mapping at larger scale, the uniform voxel representation is too memory-intensive. Therefore one typically reverts to adaptive representations using voxel hashing [805] or octrees [1027] —see Figure 7.13 and also refer to Chapter 5.

Later approaches have furthermore demonstrated scalability to larger scenes by employing a moving fusion window as in kintinuous [1180], or by including loop closures as ElasticFusion [1181] that deforms the entire dense map representation through a deformation graph.

## 7.8 Combining Vision with Other Modalities

For numerous reasons, it is advisable to combine vision with other modalities. This can provide additional robustness and precision, but it can also provide absolute scale information. Specifically, inertial sensors provide metric scale and highly reliable and robust local, relative, motion measurements. GPS/WiFi, however, may be leveraged for global (re-)localization and geo-positioning. These complementary inputs address limitations inherent to vision-only systems, making them indispensable in practical applications.

### 7.8.1 Inertial Measurement Units (IMU)

IMUs provide high-frequency measurements of angular velocity and linear acceleration, enabling precise estimation of local motion. A simple way to fuse sensory information is a loosely coupled approach where the sensory information from each sensor is independently aggregated into a pose estimate and subsequently, respective estimates are fused with a Kalman filter. While this often works fairly well in practice, for example for autonomous navigation of quadrotors [306] or nano-copters [291], it is less precise than a tightly coupled integration of sensory information.

First tightly coupled approaches leverage an EKF scheme, whereby the IMU kinematics are integrated in the prediction step, and visual keypoint measurements

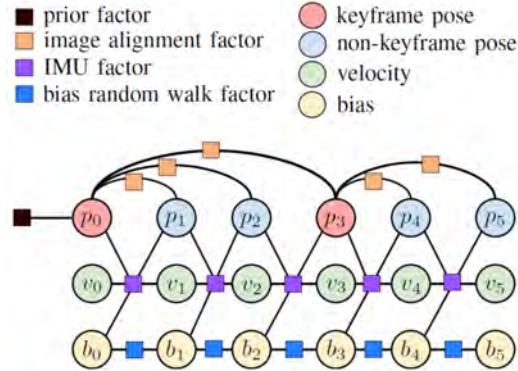


Figure 7.14 The integration of multiple sensors can be elegantly performed in tightly coupled manner using factor graphs. This is an example factor graph for tightly coupled fusion of vision and IMU [1113]. It significantly increases precision and robustness of camera motion and 3D structure —see Figure 7.15. (©2016 IEEE)



Figure 7.15 Tight fusion of the IMU measurements with direct image alignment results in more accurate position tracking (left) compared to the purely visual odometry system that only relies on image alignment (right). The fusion was achieved with the factor graph shown in Figure 7.14. The reconstructed pointclouds come from pure odometry, no loop closures were enforced [1113]. (©2016 IEEE)

serve as updates –as in the seminal work of the MSCKF [774]. Alternatively, sliding-window and batch optimizers may be formulated adopting the factor graph approach discussed in detail in Chapter 11. Figure 7.14 shows a factor graph proposed for stereo-inertial odometry [1113] that combines stereo LSD-SLAM with IMU information. Figure 7.15 shows how the tightly coupled IMU information reduces the drift leading to more crisp and precise reconstructions. This tight coupling by factor graphs is also employed by state-of-the-art keypoint-based approaches, such as ORB-SLAM3 [142], OKVIS2 [642], and many more.

*visual-inertial odometry*

In practice, some of the most accurate visual-inertial odometry systems start with inertial integration as basis, and use vision primarily to correct for the fast-growing drift owing to (doubly) integrated noise and biases over time. In addition, IMUs allow to observe metric scale of motion, as well as the sensor’s orientation

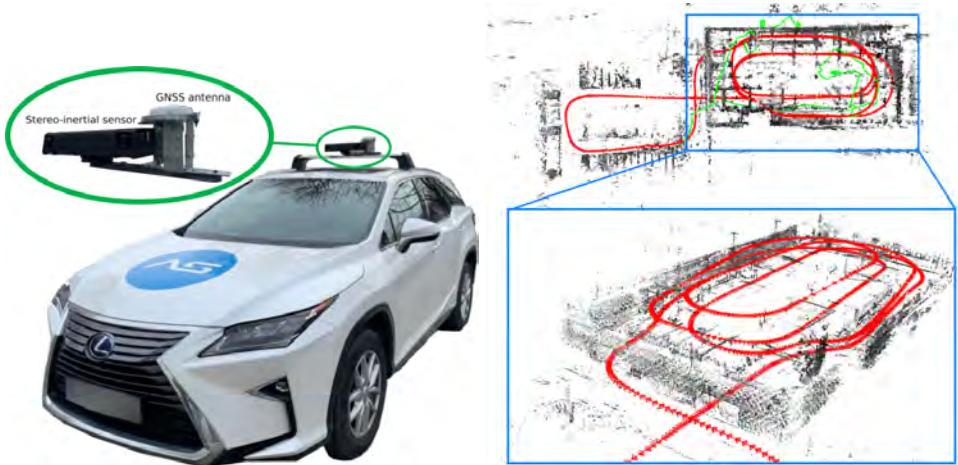


Figure 7.16 By combining multiple sensors including stereo cameras, IMU and RTK-GPS information (left), fused in a tightly coupled manner using factor graphs, one can obtain highly accurate and robust trajectories and pointclouds, both indoor and outdoor as shown in the reconstruction of a car driving on multiple levels of a parking garage (right) [1178]. (©2020 Springer)

with respect to gravity—thus reducing the gauge freedom of the system to only 4 unknowns (global  $x$ ,  $y$ ,  $z$  position, as well as yaw)—compared to 7 unknowns (global  $x$ ,  $y$ ,  $z$ , roll, pitch, yaw, as well as scale) for vision-only systems. In many ways, IMUs are complementary to vision as a modality, and thus are ideal to combine in visual-inertial SLAM or Odometry systems.

A multitude of visual-inertial odometry/SLAM methods have been proposed over the years, often as extensions of existing visual SLAM systems. Popular approaches include a visual-inertial version of ORB-SLAM [783], Direct Sparse Visual-Inertial Odometry [1133], VINS-Mono [893], BASALT [1115] and DM-VIO [1132]. The latter approach is a mono-inertial formulation that makes use of the concept of *delayed marginalization* to better capture the observability of motion in the respective sensors.

*visual-inertial SLAM*

### 7.8.2 GPS and WiFi for Global Localization

While IMUs improve local motion estimation, GPS and WiFi are essential for global localization, especially in large-scale environments. In outdoor environments, GPS provides absolute position data, enabling the system to anchor the SLAM map to global coordinates. This is critical for outdoor applications like autonomous vehicles [1178, 193] —see Figure 7.16. In indoor scenarios, WiFi signals enable

coarse localization where GPS signals are unavailable, complementing visual map-based localization.

### 7.9 Further Readings & Recent Trends

Visual SLAM is an extremely active and dynamic field of research. While we tried to provide a comprehensive overview of classical visual SLAM methods, we invariably only covered a fraction of relevant work in this chapter, and we hope the reader may look into some of the many cited works for a more in-depth coverage.

**Keypoint Detection and Matching.** With the unprecedented invariance and adaptability of learning-based keypoints over traditional handcrafted counterparts in specific settings, deep-learning-based techniques for keypoint extraction have been transforming the field. This shift has not only enhanced keypoint detection and description but has also driven significant advancements in keypoint matching techniques. Rather than relying on manually designed heuristics for descriptor matching, as in classical methods, learning-based approaches now incorporate global spatial awareness to establish correspondences more robustly. In particular, techniques such as SuperGlue [968] and MASt3R [640] leverage Graph Neural Networks (GNNs) and Transformers to refine matches by considering the broader image structure. These methods address fundamental limitations of traditional matchers by adapting to extreme viewpoint changes, illumination variations, and occlusions, where conventional local descriptor comparisons typically struggle.

SuperGlue enhances feature matching by integrating self-attention and cross-attention mechanisms, allowing it to resolve ambiguities caused by repetitive textures and occlusions. Unlike traditional keypoint matchers that operate on local descriptors alone, SuperGlue incorporates contextual information, significantly improving accuracy in both indoor and outdoor environments. Its optimal matching layer further ensures that keypoint correspondences are established robustly while allowing unmatched keypoints when necessary, making it highly effective for real-world scenarios. Similarly, MASt3R extends this idea into 3D-aware feature matching, reconstructing a 3D scene representation from two images to improve performance in textureless regions and under extreme viewpoint changes. Built upon this, MASt3R-SLAM [786] integrates these advancements into a SLAM pipeline, improving camera pose estimation, global map consistency, and loop closure strategies. By transitioning from handcrafted 2D feature matching to learning-based, context-aware, and 3D-informed approaches, these methods mark a paradigm shift in visual SLAM, enhancing scene understanding, tracking robustness, and long-term stability in complex environments.

Overall, the evolution of keypoint extraction for image matching and visual SLAM has been remarkable from traditional hand-crafted methods to deep-learning-based approaches that promise greater robustness in challenging conditions. While classical methods remain very relevant to date due to their efficiency and inter-

preability, they struggle in scenarios with textureless surfaces, extreme viewpoint changes, and illumination variations. Deep learning has addressed these limitations, driving research towards more adaptive and context-aware keypoint detection and matching techniques. However, challenges remain in balancing computational efficiency with real-time performance, particularly on resource-constrained platforms, and ensuring the availability of diverse, unbiased training datasets. Future research will likely focus on optimizing these techniques to maximize both accuracy and efficiency, making them more viable for large-scale real-world applications.

**Other Frontiers.** We emphasize that there have been many exciting developments in visual SLAM in the recent past. Among other developments, there are generalizations of visual SLAM to dynamic environments with moving and potentially deformable objects. In addition, learning-based approaches to visual SLAM are becoming increasingly popular: in a multitude of publications, more and more components of the classical visual SLAM pipeline (feature extraction, correspondence estimation, image alignment, camera tracking, BA, dense reconstruction, etc.) are being enhanced or replaced by learning-based formulations. All these ideas will be discussed in more detail in Part III of this handbook.

# 8

## LiDAR SLAM

Jens Behley, Maurice Fallon, Shibo Zhao, Giseop Kim  
Ji Zhang, Fu Zhang, and Ayoung Kim

Along with cameras, LiDAR sensors are one of the most popular sensing modalities used in robotics. LiDAR is a technology that uses a laser to actively transmit laser light pulses and then measures the time delay in those pulses reflecting off of surfaces and returning to a detector. In doing so it directly measures the distance to those surfaces. A LiDAR sensor can be used to perceive the structure of its surrounding environment and also to estimate the motion or ego-location of the sensor.

The development of LiDAR technology began in the 1960s and 1970s with stationary systems primarily used for applications such as atmospheric research, topographic mapping, and military applications. These early LiDAR systems were bulky and expensive, making them unsuitable for mobile applications. In the 1980s, advancements in laser technology and computing power allowed for more compact and affordable LiDAR systems. These systems were still primarily stationary and used in applications like terrain mapping and environmental monitoring [248, 680]. In the 1990s, the integration of LiDAR sensors with Global Positioning System (GPS) and IMUs began to enable the first mobile LiDAR mapping systems. These systems were often mounted on vehicles or aircraft to create detailed 3D maps of large areas [499]. In the late 1990s, researchers began exploring the use of LiDAR for real-time SLAM in robotics. 2D LiDARs were instrumental in driving and catalyzing progress in probabilistic mapping and SLAM in the early 2000s, and 3D LiDARs became even more prominent in robotics and self-driving vehicles with the DARPA Grand Challenge in 2004 and the DARPA Urban Challenge in 2007.

In this chapter, we start by briefly reviewing the underlying technology within different types of LiDAR sensors (Section 8.1). Then, we discuss LiDAR odometry (Section 8.2) and LiDAR-based place recognition (Section 8.3). Moreover, we discuss the integration of odometry and place recognition into full LiDAR-SLAM systems (Section 8.4). Finally, we provide pointers to further readings and discuss new trends (Section 8.5).

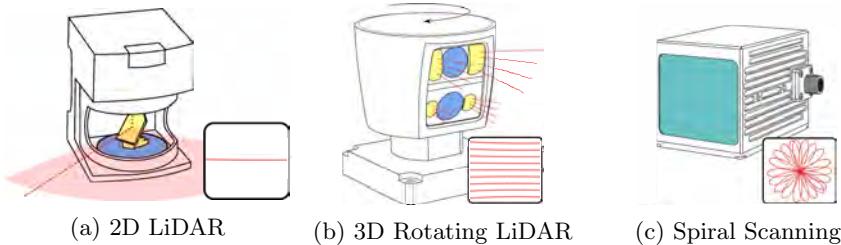


Figure 8.1 Common LiDAR sensor types and their beam patterns. (a) A standard 2D LiDAR sensor with a rotating transmitter mirror (yellow). The encoder disk (blue) is used to measure the rotation angle of the mirror. (b) An example of a mechanical multi-beam LiDAR emitting multiple laser beams from the different emitters (yellow), which are then detected using the detectors (blue). The entire sensor head rotates to generate a 360° horizontal field of view. (c) A macroscopic steered Risley prism LiDAR with an spiral scanning pattern which only measures in the direction of the lens window (cyan). Due to the spiral pattern it can produce denser point clouds over time.

## 8.1 LiDAR Sensing Preliminaries and Categorization

By rotating the laser emitter and detector within a LiDAR sensor in one or two axes, it is possible to build up a detailed point cloud of the environment around the sensor. The most basic principle is time-of-flight (TOF), which employs laser pulses to infer distances using the measured time it takes for emitted light pulses to return to the detector. TOF LiDAR sensors can capture high-resolution measurements but are sensitive to external light, which reduces the signal-to-noise ratio (SNR) [597] and therefore the accuracy and frequency of measurements. Besides TOF, the techniques of Amplitude Modulated Continuous Wave (AMCW) and Frequency Modulated Continuous Wave (FMCW), originally developed for radar sensors have been adopted for LiDARs.

The classes of LiDAR sensors are categorized by their sensing mechanisms [932] and can be classified into mechanical LiDARs, scanning solid-state LiDARs, flash LiDARs, and sensors using macroscopic scanning Risley prisms. Among these, we will examine two commonly utilized types: 2D/3D mechanical LiDARs and macroscopic scanning LiDARs.

Mechanical LiDARs are the most common category of LiDAR. They use a rotating assembly to direct the laser beam but face limitations due to mechanical wear and lower rates of data acquisition. The simplest 2D mechanical LiDAR sensors use a rotating mirror to direct a single laser beam and to measure distances, as shown in Figure 8.1a. By using an encoder disk, the angular orientation of the LiDAR beam can be measured and paired with the range measurement to produce a 2D profile measurement. By its nature, this LiDAR can only scan in an individual 2D plane.

The demand for a single sensor that can scan in full 3D was motivated by the

*mechanical LiDAR*

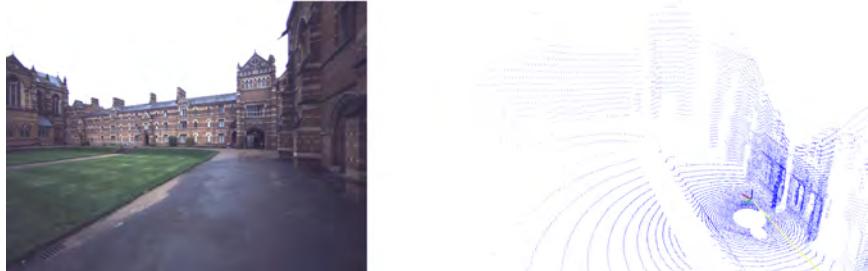


Figure 8.2 Example of a single multi-beam LiDAR scan and a corresponding image. The scan is from a 64-beam Hesai QT64 scanner with 104° vertical field-of-view. Note the individual scanning lines and also how dense the point cloud is close to the device (the colored axis in the 3D view) and much sparser the cloud is further away.

DARPA Grand Challenges (an early self-driving car competition) in the 2000’s and led to the development of the pioneering Velodyne HDL-64E sensor. It was used by most of the teams [1112, 769, 535] in the challenge. Within a 3D mechanical LiDAR, multiple laser emitters are mounted on a single rotating mechanism to capture individual range measurements pointing in different elevation angles as the entire mechanism rotates through 360 degrees in the azimuthal axis as illustrated in Figure 8.1(b). The resulting point cloud can capture a highly detailed 3D depiction of objects and the sensor’s surroundings as shown in Figure 8.2, as discussed in various studies [1175, 556, 932]. Subsequently, the technology has evolved —with the size and price of LiDAR sensors dropping significantly.

#### *solid-state LiDAR*

A wider variety of prototype sensors have emerged more recently drawing on a variety of physical properties including solid-state LiDARs, Risley prisms, and polygonal mirrors. In contrast with traditional scanning laser, many of these sensors use Micro-electromechanical (MEMS) [467] mirror technology or optical phased arrays (OPA) [446] to avoid, or at least minimize, mechanical rotation. This is important because it can enhance the LiDAR’s lifespan and reliability in environmental mapping, since fewer mechanical parts need to be actively actuated which reduce the mechanical wear.

A notable advancement is the use of Risley prisms [665] which enables rapid, controlled beam steering with much smaller amount of physical movement. This innovation results in a more compact sensor, albeit with more limited FOV currently.

#### *remission*

All the aforementioned sensors produce sets of individual range measurements with intensity (often also called remission) value for each measurement, *i.e.*, how much of the LiDAR beam is reflected. The range measurements with associated angles of the individual beams can be then converted into a 2D or 3D point cloud. Yet, more recent advances have introduced additional measurement capabilities beyond the default range measurement. For instance, FMCW LiDARs continuously project light with varying frequency and can measure the relative velocity of the

object the beam hits using detected frequency shifts. This is similar to how FMCW is used in radar. This approach is useful in dynamic environments or challenging scenarios [1198], although it tends to be more complex and costly compared to other variants. Another innovative sensing mechanism is flash LiDAR, which can provide ambient channels resembling photometric measurements, similar to those obtained by cameras. These technologies, with their different characteristics of power consumption, weight and cost, provide a variety of options when carrying out LiDAR odometry and SLAM for different applications.

## 8.2 LiDAR Odometry

The first building block of LiDAR SLAM is LiDAR odometry. The goal of LiDAR odometry is to estimate the incremental ego-motion of a robot or vehicle in real-time given a LiDAR scan and past observations, *i.e.*, a single scan or multiple scans aggregated into a local map. Here, the term *scan* refers to a single sweep or cycle of data collected by the LiDAR sensor. More specifically, a scan typically represents one complete rotation or one full sweep of the sensor providing a contextual snapshot of the surrounding environment at a specific time. Thus, scans are often time-stamped, allowing them to be ordered and processed as sequential observations.

At the heart of LiDAR odometry lies the technique of scan registration, also referred to as *scan matching*. Scan registration involves finely aligning a pair of scans to estimate the precise relative transformation between the scans. A scan is effectively a set of points or a point cloud. In the literature, several important algorithms have been proposed for point cloud registration, including Horn’s method [477] and ICP. Although Horn’s method provides a closed-form solution, it requires pre-determined correspondences, which limits its applicability in real-world scenarios. In contrast, the ICP algorithm [79, 956] does not require prior knowledge of correspondences and has therefore become a more popular and fundamental technique for determining the relative transformation between point clouds. We will discuss ICP further in the next sections.

The original development of LiDAR SLAM can be traced back to the seminal work of Lu and Milios [698], who pioneered the concept of globally consistent 2D range scan registration by introducing the idea of a network of poses —a concept closely resembling the modern pose-graph approach. This work also laid the groundwork for LiDAR odometry by defining the fundamentals of 2D scan registration. Key contributions to the probabilistic framing of scan-to-scan matching were made by works including [823, 595]. While points and lines are common choices in 2D scan matching, Olson [822] introduced an impactful approach using correlation techniques for real-time registration.

Early extensions to 3D LiDAR built on these 2D scanning techniques by actively moving the sensor in a nodding [433] or rotating manner [100], or by passively

using the motion of a human carrier [102] or vehicle [100] to accumulate denser 3D point clouds. These 3D point clouds enabled 3D scan matching for LiDAR SLAM but introduced significant computational challenges due to the increased data size. Addressing these challenges, LOAM [1266] demonstrated real-time scan matching capabilities forming the basis for follow-up methods in LiDAR odometry and SLAM.

### 8.2.1 Foundations of Scan Registration

*scan registration*

Scan registration is a fundamental component in LiDAR odometry and mapping systems. It involves the alignment of two scans to achieve an accurate alignment and mapping. The goal is to find the transformation, *i.e.*, rotation  $\mathbf{R} \in \text{SO}(3)$  and translation  $\mathbf{t} \in \mathbb{R}^3$ , that can best bring one of the scans (potentially a recent scan from a sensor) into alignment with the other (*e.g.*, a previous scan or a local map). In doing so, this process also yields the relative position from which the scans were taken. A large body of techniques and algorithms have been developed to perform scan registration with high accuracy, robustness, and low computational cost.

**Iterative Closest Point and Its Variants.** As introduced in Chapter 5, a point cloud is defined to be a set of points in a three-dimensional coordinate system, represented mathematically as  $\mathcal{P} = \{\mathbf{p}_i \in \mathbb{R}^3 \mid i = 1, 2, \dots, N\}$ , where each  $\mathbf{p}_i = (x_i, y_i, z_i)$  denotes the 3D coordinates of a point. For scan registration, the ICP algorithm [79] minimizes the total registration error between two point clouds  $\mathcal{P}$  and  $\mathcal{Q}$ . Let us denote  $\mathcal{P}$  as a source and  $\mathcal{Q}$  as a target point cloud.

ICP iteratively determines transformations  $\mathbf{R}^k, \mathbf{t}^k$  for an optimization iteration  $k$  that minimize the total registration error, which is measured by different distance metrics  $d$  and is given by

$$\mathbf{R}^k, \mathbf{t}^k = \arg \min_{\mathbf{R}, \mathbf{t}} \sum_{(\mathbf{p}, \mathbf{q}) \in \mathcal{C}} d(\mathbf{p}_i, \mathbf{R}\mathbf{q}_i + \mathbf{t}), \quad (8.1)$$

where the set of correspondences  $\mathcal{C}$  between the source point cloud  $\mathcal{P}$  and target point cloud  $\mathcal{Q}$  is given by

$$\mathcal{C} = \{(\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in \mathcal{P}, \mathbf{q} \in \mathcal{Q}\}. \quad (8.2)$$

In the ICP algorithm, determining the transformation between the source and target is achieved iteratively by recomputing for each iteration  $k$  a new set of correspondences  $\mathcal{C}$  based on the last transformation at iteration  $k - 1$  given by rotation  $\mathbf{R}^{k-1}$  and translation  $\mathbf{t}^{k-1}$ .

To minimize the total registration error in (8.1), two aspects must be specified:

- 1 What geometric relation is used to define the distance measure  $d(\cdot)$ ? The aim is to align two point clouds as tightly as possible, and this *tightness* cannot be determined without defining a distance metric.

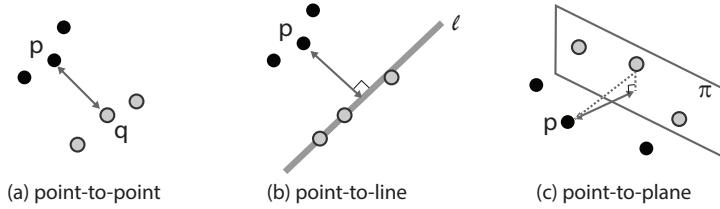


Figure 8.3 Typical distance metrics used in ICP. (a) Point-to-point distance is as straightforward as the Euclidean distance between two points. (b) and (c) The point-to-higher-level feature (*e.g.*, line or plane) is calculated as the shortest distance to the reconstructed line or plane using the target points.

- 2 How is the correspondence set  $C$ , used for minimization, determined? This involves identifying the corresponding target point  $q \in Q$  for each source point  $p \in P$ .

Common approaches used to address these two aspects are discussed below.

#### 8.2.1.1 Distance Measure in Registration Residual

The first question above involves deciding which geometric elements to use for defining the residual, possibly going beyond simple point-to-point distances. Typically, points, lines, and planes are the most common geometric elements used to define the distance in equation (8.1), as summarized in Figure 8.3.

Point-to-point ICP is the most basic approach and it minimizes the Euclidean distance between corresponding points in the two point clouds. Pioneering works in ICP by Zhang [1278] and Besl and McKay [79] formulated shape (*e.g.*, curves and surfaces) matching as a point matching problem by representing shapes as sets of points. This point-to-point cost is straightforward and simple, but can be sensitive to noise, outliers, and sparsity of the measured points. Distances to lines are also commonly used as an error measure. The point-to-line distance measures points in one point cloud and lines (formed by connecting points) in the other point cloud. It can provide better results in structured environments with linear features. By exploiting higher-level geometric features we can go further. We can measure the distance between points in one point cloud and planes (local surfaces) in the other point cloud. This approach is more robust to noise and can achieve higher accuracy in environments with planar surfaces.

*Point-to-point distance*

*point-to-line distance*

*point-to-plane distance*

Extending from these basic geometries, other ICP variants introduce different distance metrics. Techniques which employ multi-distance metrics [836], continuous-time formulation [256], and adaptive thresholds [1130] are more some of the more recent ICP advances. Other methods [982, 590] opted to evaluate differences in a probability distribution of a local neighborhood than using Euclidean distances.

Another well-known distribution-based matching is the normal distributions trans-

form (NDT) [82]. NDT divides an input point cloud into a set of voxels and fits a normal distribution to the points in each voxel (see Chapter 5.3.2.2 for more details). Instead of incurring the cost of determining nearest neighbor associations, it takes advantage of voxelization to carry out a distribution-to-distribution matching process. This process can take advantage of a smoother and more robust registration cost surface, especially in complex environments.

#### *correspondence search*

##### 8.2.1.2 Determining Correspondences

The second core design decision of common ICP algorithms is data association or correspondence search between the source  $P$  and the target  $Q$ .

In the most basic form, determining correspondences between  $P$  and  $Q$  can be achieved geometrically by finding the nearest neighbor of a point  $\mathbf{p} \in P$  in the target  $Q$ , where we use the iteratively updated transformation  $(\mathbf{R}^{k-1}, \mathbf{t}^{k-1})$ , which is given by:

$$C = \{(\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in P, \mathbf{q} = \arg \min_{\mathbf{q}' \in Q} \|\mathbf{p} - (\mathbf{R}^{k-1}\mathbf{q}' + \mathbf{t}^{k-1})\|_2\} \quad (8.3)$$

However, this is typically an expensive operation when computed over a large point cloud with thousands of points.

To make real-time operation of LiDAR odometry possible, there are two common strategies used to reduce the time for correspondence search: (1) reducing the set of potential candidates for a correspondence, or (2) employing a different search strategy than exact distance-based neighbor search to more quickly find potential candidates.

Several ICP variants used in popular LiDAR odometry systems [1266, 836, 996] employ the first strategy to reduce the set of potential correspondences by building maps with reduced candidate sets,  $P' \subset P$  and  $Q' \subset Q$  by determining points that fulfill certain geometric criteria. The criteria used include determining points lying on edges or surfaces [1266, 836], removing less descriptive points that correspond to the ground plane [995], or downsampling of the target scan [1130, 256]. While these strategies certainly speed up the correspondence search, they have the potential drawback of removing true correspondences from the target  $Q$ .

In contrast, the second strategy employs search structures with efficient approximations which enable faster correspondence searches even though they may not always yield the exact nearest neighbor. In this direction, a common strategy is to use projective neighbor search in range images [66] or leveraging voxel grids for approximate neighbor search [1266, 256, 1130]. Furthermore, while we covered here the pure geometric correspondence search in Euclidean space, it is also possible to use alternative distance metrics, as well as to apply projections into a feature space [836] to identify correspondences.

In the following sections, we will discuss other components of a LiDAR odometry system that are integrated around the core ICP component to build out a complete

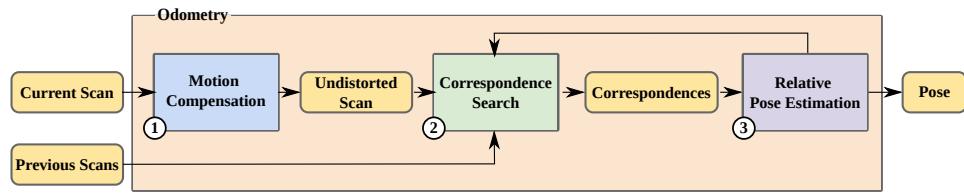


Figure 8.4 Components of a LiDAR odometry pipeline: Given a current scan, (1) motion compensation accounts for the motion of the sensor during the scan process resulting in an undistorted scan. Then, (2) correspondences between the undistorted scan and the previous scans, either a single scan or aggregated scans in a local map, are determined. Finally, (3) the relative pose estimate is determined via a scan registration to estimate the relative pose of the current scan. These steps are iterative with the correspondence set refined based on the intermediate relative pose estimates. After convergence, the final pose estimate is the output of the LiDAR odometry system.

scan registration system which can align a sequence of scan observations so as to estimate the relative motion of a robot.

### 8.2.2 Common Components for LiDAR Odometry

This section outlines the key modules involved in a LiDAR odometry system: point cloud motion compensation, identifying correspondences, and pose estimation via scan registration. Point cloud motion compensation addresses the distortion caused by the motion of the LiDAR during scan acquisition. Correspondence search identifies matching points between consecutive scans that are useful for matching and scan registration. Finally, the pose estimation module estimates the sensor's motion since the previous registration. Registration can be carried out either *scan-to-scan* between consecutive scans or *scan-to-map* with respect to a local map. Together, these modules ensure accurate and reliable LiDAR odometry. Figure 8.4 shows the interplay between the different components in a common LiDAR odometry pipeline.

#### 8.2.2.1 Point Cloud Motion Distortion Compensation

Motion distortion in LiDAR odometry occurs because a LiDAR sensor captures a scan over a period of continuous motion.<sup>1</sup> Due to this movement during the scan period, the sensor will emit laser pulses and receive ranging returns from slightly different times and positions. This means that a single scan does not represent a static snapshot of the surroundings at a single position but instead a set of points each captured from a slightly different scanning position. This continuous movement during the scanning process will lead to inaccuracies and in turn a distorted point

*scan-to-scan odometry*  
*scan-to-map odometry*

<sup>1</sup> In modern LiDAR SLAM, LiDAR sensors are often mounted on moving vehicles, robots, or wearable devices. Assuming a scan rate of 10 Hz, the scan period will then be 0.1 seconds.

cloud if left uncorrected.<sup>2</sup> For example, Figure 8.5 clearly illustrates a distorted point cloud sample and highlights the need for proper motion compensation. As can be seen, undistorting the point cloud to account for the motion of the LiDAR sensors is an important pre-processing step in LiDAR odometry which can improve accuracy and robustness. Compensation methods have been developed to correct this effect including a constant-velocity model, continuous-time trajectory optimization, and using IMU measurements.

**Constant-Velocity Model.** The constant velocity model assumes that the robot maintains the same translational and rotational velocities estimated during the previous time step. As this model does not require any additional sensors, it can be widely used in simpler LiDAR odometry systems [256, 1130], however, the constant velocity assumption is inherently less accurate when the motion contains high-frequency motions.

**Continuous-Time Trajectory Optimization.** Another widely used approach for motion compensation is continuous-time trajectory optimization techniques using splines [282, 712] and the GP [50] (*cf.* Chapter 2). Continuous-time trajectories allow pose estimates to be made at any time instant without relying on linear interpolation. They can be used to remove the distortion of each individual point, however, conventional continuous-time trajectory optimization is time-consuming and often implemented offline [712].

**IMU-based Motion Compensation.** IMU measurements directly measure high-frequency motion with gyroscopes and accelerometers. They can be integrated over the scan period as an effective approach for motion compensation [1266, 996]. IMU-based motion compensation pre-integrates the LiDAR pose using the most recent IMU data and then uses that predicted trajectory to rectify for point distortion. Thanks to the high frequency of IMU measurements (*e.g.*, 200 Hz), IMU-based motion compensation is highly effective for jerky robot motions and is now the *de-facto* standard for most robot platforms. Nonetheless, this method needs to be used carefully because IMU measurement noise, bias, estimation errors, and poor clock synchronization can cause this approach to under-perform other simpler methods.

**Point-wise Registration.** Being firstly proposed in Point-LIO [442], this point-wise approach is fundamentally different from existing scan-based LiDAR odometry frameworks. In this framework, the state is updated by processing each LiDAR point when it is received, rather than accumulating a complete scan. As a result of this design, the proposed method does not suffer from intra-scan motion distortion.

#### 8.2.2.2 Feature-based LiDAR Odometry

Once motion distortion is corrected, point correspondences can be established. Similar to visual SLAM (see Chapter 7), leveraging features for scan association is well studied. A feature-based approach allows efficient representation of the scan by

<sup>2</sup> A similar type of distortion effect during camera image capture and is known as the rolling shutter effect.

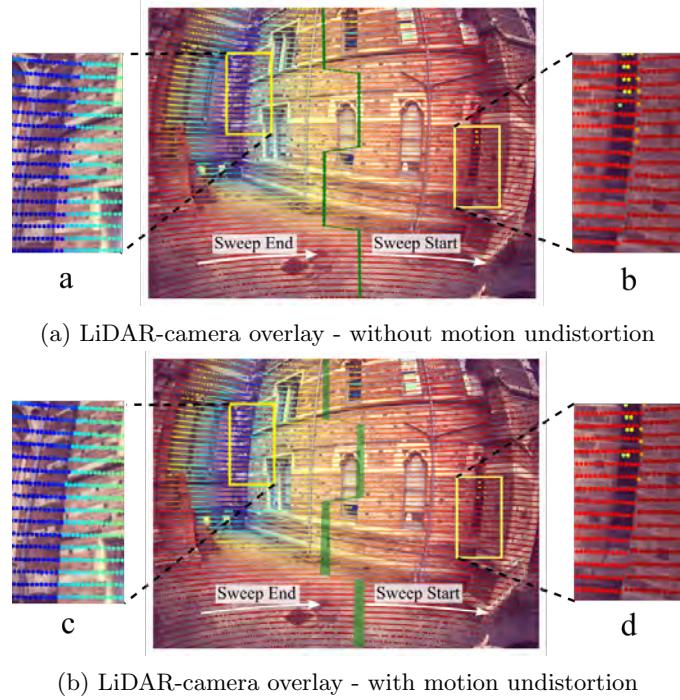


Figure 8.5 LiDAR point cloud overlaid on a camera image. A slight misalignment at the start of the sweep ('b') worsens significantly by the end of the LiDAR sweep ('a'). At the bottom, more consistent overlays are observed for both the start ('d') and end ('c') after motion compensation. From [1069].

processing only a small number of features extracted from the point cloud. Correspondence matching and residual computation can also be performed at the feature level, substantially reducing the overall computational cost.

**Low-level Features.** Lines and planes are the most commonly used features in practice. In this line of study, the well-known LOAM algorithm [1266] was a significant breakthrough in LiDAR SLAM. LOAM uses a low-level detector to efficiently identify mid-level features that can contribute to scan registration. Points with high or low curvature are identified to detect edges and planes. The curvature of each point is calculated by analyzing the differences between the point and its neighbors. High curvature points are marked as edge features, while low curvature points are identified as planar features. Not all points are used as features; the algorithm selects a subset of the most significant edge and plane points to reduce computational complexity while maintaining accuracy.

Principal component analysis can also be employed to identify the principal directions of a local neighborhood of a point for effective feature detection [67, 750]. By calculating the eigenvalues and the corresponding eigenvectors of the covariance

matrix of a point and its neighbors, the main axes of variation can be determined. This analysis allows us to discern the geometric properties of the points. Points with one dominant eigenvalue can then be classified as edge points, indicating sharp transitions or boundaries. In contrast, points with two similar eigenvalues can be identified as being from planar regions, representing flat surfaces within the point cloud. This enables differentiation between edge and planar features, enhancing the accuracy of LiDAR odometry by considering the geometric properties.

**High-level Features.** In LiDAR odometry, high-level features such as semantic, surfel, and intensity features can also play an important role in enhancing the accuracy and robustness of the system. These features provide richer information about the environment compared to low-level features, facilitating better scene understanding and more accurate mapping.

- **Semantic Features:** Semantic features involve the use of machine learning and deep learning techniques to classify and label the LiDAR point cloud into object categories such as vehicles, pedestrians, buildings and vegetation —typically to distinguish between dynamic and static objects. It can improve the reliability of odometry by focusing on stable landmarks [187].
- **Surfels Features:** Surfels are small disk-like representations of the surface geometry of a point cloud (see Chapter 5.3.2.2). An ellipsoid disk can be fit to a set of points with the ellipsoid’s principal semi-axes lengths determined by the eigenvalues of the covariance matrix of nearby points. This type of surface representation can then be used to compute point-to-plane distances during scan registration [843, 66, 900].
- **Intensity Features:** Intensity features [277, 415] refer to the reflectivity or intensity values of the LiDAR returns. These values provide additional information about the material properties and surface characteristics of objects in the environment. They improve the robustness of feature matching by providing an additional dimension of information, which can be crucial in challenging scenarios such as structure-less environments.

#### *8.2.2.3 Direct Point-wise LiDAR Odometry*

A problem with the feature-based approach described above, is that it tends to discard subtle contributions from isolated points which do not clearly correspond to planes or edges. This is particularly a problem when mapping unstructured environment with bushes or branches in natural environments. It also requires tuning of hand-engineered feature detectors when moving from one sensor to another. Additionally, the number of points to be processed by the backend typically scales linearly with the size of the LiDAR scan. The computational cost of this approach can become impractical when working with modern 64 or 128 beam sensors.

Alternatively, one can use the points directly —without extracting mid-level features. Similar to the direct methods in visual SLAM, we can directly align points

in an ICP-like manner. However, due to the high computational cost during point-wise correspondence matching, this direct methods were not favored in the early development of LiDAR SLAM.

Paving the way for direct methods, Zhou et al. [1292] made it practical to use direct methods by speeding up the nearest neighbor search with a GPU-accelerated KD-tree implementation. Later the direct method Fast-LIO2 by Xu et al. [1208] demonstrated highly accurate frame-rate odometry without suffering from a correspondence search bottleneck when the map grows large using an extension, the so-called incremental KD-tree (iKD-tree). The iKD-tree can adapt to the distribution of points by occasionally rebalancing itself to allow for efficient add, remove and query operations, which avoids rebuilding the tree for each added scan.

*iKD-tree*

#### 8.2.2.4 Local Mapping and Pose Estimation

Once reliable correspondences have been obtained, the next step is to achieve consistent registration of the consecutive scans. As mentioned before, incremental pose estimation in LiDAR odometry is achieved, in most cases, via a scan registration with a variant of ICP (see Section 8.2.1).

Initially methods focused on *scan-to-scan odometry* and sought to achieve full frequency (10 Hz) output while treating each consecutive scan registration operation as being independent. However, as individual LiDAR scans can be relatively sparse, many points will have unsuitable correspondences if the reference scan is simply the previous scan. This will result in registration errors accumulating and an inconsistent overall map.

A more modern approach is to build a detailed and accurate local map around the sensor which is known as *scan-to-map odometry*. This paradigm has been successfully used in 2D LiDAR SLAM [458], 3D LiDAR odometry [1208, 256, 1130], and 3D LiDAR SLAM systems [836, 66, 843] and has been seen to reduce overall drift rates.

The motion prediction from either scan-to-scan odometry or IMU pre-integration can be used to pre-align the incoming scan before a fine registration to the persistent local map is carried out. The local map which becomes much dense than an individual scan results in much more suitable inlier associations. After registration, the incoming scan will be added to this local map.

Earlier LiDAR odometry approaches [1266] needed to resort to interleaving scan-to-scan odometry at a high frequency with scan-to-local-map odometry at lower frequency due to compute restrictions. More modern LiDAR odometry approaches [256, 1130] now use a single-stage scan-to-map alignment with direct point-wise correspondences enabled by a voxelized local map representation.

Finally, a quite different approach to LiDAR odometry is to use deep learning. Efforts to learn ego-motion directly from LiDAR measurements has resulted in some promising works. Early studies employed supervised learning using ground-truth labels [653], and this line of work has been extended to unsupervised learning meth-

*scan-to-scan odometry**scan-to-map odometry*



Figure 8.6 Despite the large visual differences between Place A and Place B in the RGB images (aerial view and robot’s front-looking view), their corresponding LiDAR scans exhibit structurally similar patterns. This causes structural perceptual aliasing, where distinct places appear similar to LiDAR-based place recognition algorithms due to shared road topology and surrounding structures. This example is captured from the SNU Afternoon sequence of the STheReO dataset [1259].

ods [210]. While the performance of deep LiDAR methods are generally promising, concerns have been raised regarding their generalization capabilities.

### 8.2.3 Summary of LiDAR Odometry

To summarize, common 3D LiDAR odometry algorithms can produce highly accurate and robust motion estimates by iteratively aligning incoming scans to a running local map—often with the support of IMU measurements or motion models to correct for motion distortion of the scan. Resulting systems can achieve drift rates in the order of 1 m per 1000 m traveled—but the performance is highly dependent on the environment around the robot and the level of dynamics present in the scene and the dynamics of the sensor itself. Accounting for this remaining amount of small drift is a key aspect of LiDAR SLAM with place recognition being a key component of such a system.

## 8.3 LiDAR Place Recognition

*place recognition*

Place recognition systems seek to identify places that have been previously visited by a robot/sensor. It is a key capability in LiDAR SLAM and also enables related problems, including multi-session SLAM as well as global localization in a prior map. Unlike visual data, LiDAR data allows a robot to obtain consistent metric 3D information about the surrounding environment. This capability ensures that LiDAR is less affected by lighting condition changes than conventional cam-

eras. However, despite this advantage over visual localization, the nature of LiDAR sensing presents unique challenges for LiDAR place recognition. For example:

**Sparse Data.** In contrast to visual measurements, where pixels are dense (*e.g.*, megapixel cameras) and organized (*i.e.*, structured into a regular 2D grid), the spacing and local density of points captured by conventional LiDAR sensors varies depending on the type of sensor (*e.g.*, the number of beams) and the sensing range (*e.g.*, points farther away are sparser). The resolution of point clouds is typically much lower than camera images. Because of these limitations, LiDAR place recognition typically does not rely on local keypoint descriptors, where each point has its own feature descriptor. To address the lack of structure, approaches identify semantically meaningful point cloud segments [285, 1271] or compute global descriptions [567, 1209] (*i.e.*, a single representative descriptor for a scan). Recently, with the advancement of deep learning, learning to determine robust local keypoint descriptors has also been actively studied [156]. Seminal papers and different paradigms in the area of LiDAR-based place recognition will be revisited in more detail in Section 8.3.2.

**Structural aliasing.** The second difficulty that LiDAR-based place recognition systems face is structural repetition in man-made environments such as long corridors or indistinguishable structures on highways. Consider the corridors on each floor of a regular modern office building. Using a camera, visual place recognition might be able to identify unique or descriptive visual texture (*e.g.*, pictures, posters or decorations) on otherwise identical corridor walls. However, it is very difficult to distinguish the specific floor using only a single scan obtained in the corridor. A similar challenge arises in outdoor environments, where perceptual aliasing can occur between visually distinct places that share similar structural layouts in LiDAR scans, as shown in Figure 8.6. Global LiDAR descriptors such as ScanContext [567] typically fail in such situations. Other approaches using object-level clusters, such as SegMap [285] and InstaLoc [1271], have been developed using higher level semantic features and can be more successful in such situations.

In summary, research needs to keep in mind these specific challenges when developing LiDAR place recognition methods.

### 8.3.1 Problem Definition

In this section, we will focus on the task of place recognition —the loop closure candidate detection problem. Given a query (*i.e.*, a scan represented as a point cloud), the objective is to retrieve corresponding entries from a database that are similar to the query. The database (*i.e.*, the previously visited places) is a set of disjoint place descriptors spatio-temporally acquired in an explored region.

A key consideration in LiDAR place recognition is the robustness of the retrieval method to variations in the sensor type, acquisition time, and robot pose. For example, the LiDAR type used in a query may differ from that used to create the

database if different LiDAR devices were employed. Furthermore, a temporal gap between mapping (*i.e.*, building the database of visited places) and revisiting a place at a later point in time is inevitable. This temporal gap might lead to structural changes in the environment as well as differences due to dynamics caused by moving objects or people. However, the most significant variation arises due to changes in the robot’s pose. Translation and/or rotation shifts between the database and the query result in different appearance of the captured sensor data of the same environment. Consequently, LiDAR place recognition methods must be robust to pose variations and environmental changes at the same time.

If a method cannot determine pose variance but still can correctly identify a candidate, it is said to have *invariance*. If it can also estimate pose variance, it is described as having *awareness* of the revisit pose variations. Researchers have particularly focused on this awareness property for two main reasons. First, it serves as a good initial guess for fine registration, which is crucial when establishing the SE(2) or SE(3) constraints required for estimating precise loop closures (see Section 8.4.1). Secondly, working towards the more complex goal of awareness can naturally enhance the invariance capability (*e.g.*, estimating heading changes [562] or inferring the degree of overlap [188]).

### **8.3.2 Methods for LiDAR Place Recognition**

In addition to achieving invariance and awareness, we should note that point cloud representations with different levels of granularity have been proposed to address the unstructured nature of the raw LiDAR measurements and to ensure real-time place retrieval performance for large-scale robot autonomy. Approaches for descriptor-based LiDAR place recognition can generally be categorized as using either local or global descriptors for retrieval and matching in the database. That said, there are variants that as well as using descriptor distance for place recognition also directly learn a place similarity function. In the following, we will discuss these different paradigms in more detail.

#### *8.3.2.1 Local Descriptors*

In the early days of LiDAR place recognition research, and corresponding to the evolution of visual place recognition methods (*e.g.*, SIFT [696], ORB [950], DBoW2 [362]), computing local keypoint descriptors was a natural approach both for 2D [1093] and 3D LiDAR sensors [101]. However, 3D local descriptors specifically developed for dense RGB-D point cloud registration or object recognition [957, 958, 1100] typically struggle to be adapted to the sparsity and unstructured nature of LiDAR sensing —particularly in outdoor scenarios. To mitigate this sensitivity, methods have been proposed that use the statistical distribution of local keypoints (*e.g.*, histograms) [462]. However, these descriptors remain limited to a local neighborhood,

which results in poor descriptiveness due to the lack of metric structural context from across an overall scan.

#### 8.3.2.2 Global Descriptors

In contrast to the local approaches, global descriptors leverage the higher level patterns in the entire scan rather than concentrating on low-level local keypoints. These methods aim to address the lack of structure by building a simpler and coarser representation. In turn, this often results in matching methods which are more computationally efficient. Two coarse representations which have been widely used to generate global descriptors are as follows:

**Bird's-eye-view (BEV).** BEV representations transform a 3D point cloud into a structured, coarse-grained, top-down image using either a polar representation or a sparse grid representation. Scan Context++ [562, 567] and RING++ [700, 1209] are examples of approaches using this representation.<sup>3</sup> The former proposed a yaw alignment matching algorithm to achieve orientation invariance, and the latter theoretically proved its invariance and awareness by leveraging the Radon transform.

**Range images.** As an alternative to using a 3D point cloud, a range image (see Section 5.1) provides a structured, well-aligned representation of a scan. The recent advancements of dense multi-beam LiDAR technology (see Section 8.1), has made representing LiDAR data as a dense range image a much more suitable approach. The advantage here is that approaches can directly borrow well-established tools from the computer vision field such as convolutional neural networks (CNN) [606] or Vision Transformers [280] to extract a detailed feature representation. Overlap-Net [188, 717] showed that yaw invariance can be achieved by applying an overlap loss to the range image, while more recently FRAME [1025] demonstrated LiDAR place recognition in mines using range images.

#### 8.3.2.3 High-level or Combined Descriptors

To derive a descriptor for a scan (either local or global), some approaches use a hybrid strategy or even learn directly a similarity function for place recognition.

**Segmentation-based approaches.** As discussed previously, representing a scan with a single descriptor can be vulnerable to structural aliasing. Because of this attempts which describe a place using a set of meaningful objects or segments have been proposed to increase uniqueness and descriptiveness and to avoid perceptual aliasing. These methods include SegMap [285] and InstaLoc [1271].

**Descriptors which bridge between local and global.** The previously introduced global descriptors are typically effective only when the point cloud projections are consistently aligned in a specific direction (*i.e.*, top-view or spherical-view). This requirement may restrict the application of the method to vehicles traveling with

<sup>3</sup> As an example RING++ [1209] uses a representation of 120 by 120 pixels equally spaced over a  $[-80, 80]$  meter grid.

predictable directions along roads. To address this issue, BTC [1255, 1256] was proposed that utilizes both local and global descriptors. This approach aims to maintain the local geometry and the overall structure of a scan, effectively combining the strengths of each type of descriptor.

**Direct 3D Data Processing.** More recently, data-driven approaches have been proposed that can achieve retrieval using the raw unstructured points directly without handcrafted rules. In particular, deep learning-based methods [1116, 156] have been proposed to extract robust point-wise features that are resilient to the diverse sensor sparsity and local surface distributions. In particular, Cattaneo et al. [156] showed that a pipeline designed with a triplet loss for place discrimination and differentiable relative pose estimation can achieve improved registration and overall benefits LiDAR SLAM. This can also be interpreted as evidence that focusing on awareness can enhance both invariance and discriminative capabilities.

### 8.3.3 Summary of LiDAR Place Recognition

In summary, LiDAR place recognition has the same role as visual place recognition and shares common attributes and its performance is defined by the same metrics.

The sensor modalities and related techniques are often complementary as in many of the locations where LiDAR place recognition fails, visual place recognition succeeds; and vice versa. In a mobile robot navigation system, both sensor modalities are often used to ensure robust and reliable results.

In the next section, we will describe how LiDAR place recognition can be used with pose-graph optimization to correct the unavoidable drift which occurs with LiDAR odometry so as to form a consistent, accurate, and scalable LiDAR SLAM system.

## 8.4 LiDAR SLAM

The purpose of the LiDAR odometry systems described in Section 8.2 is to estimate a *locally consistent* motion of the sensor as it moves through the environment. However, this motion estimate will inevitably accumulate drift as the device travels a long distance.

*locally consistent*

To counteract this drift, a LiDAR SLAM system can maintain a *globally consistent* estimate for the entire history of the mapping operation by recognizing when the sensor returns to previously visited parts of the environment. These recognition events are known as *loop closures* and they can be used by the SLAM system to correct not just its current pose estimate but also to revise the full trajectory of previous pose estimates—as well as the corresponding map representation.

*global consistency*

*loop closure*

A key property that we seek for a LiDAR SLAM system is that it maintains a globally consistent map. This requires the system to achieve consistency between

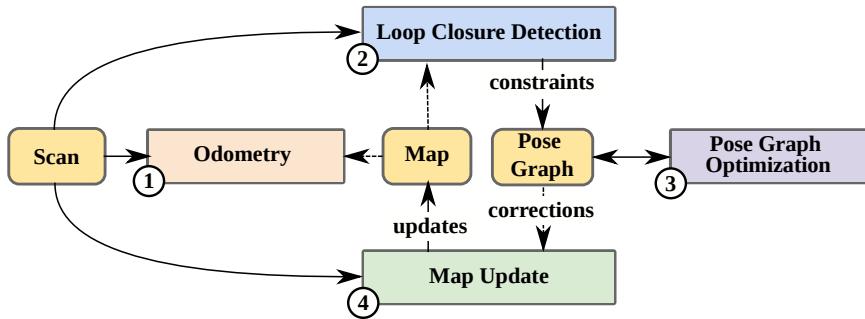


Figure 8.7 Conceptual structure of a typical LiDAR SLAM system composed of multiple components: (1) Odometry estimates the robot/sensor pose, (2) Loop Closure Detection determines if a place has been revisited, (3) Pose-graph optimization uses the loop closure constraints to correct the pose trajectory using factor graph optimization, and (4) Map Update uses the most recent pose trajectory to revise the map representation.

our current observations and past observations in a single map representation. This task is particularly challenging in large-scale and potentially dynamic environments.

In Section 8.2, we discussed the development of approaches for LiDAR odometry which can achieve drift rates as low as one meter per kilometer traveled and then in Section 8.3 we reviewed different methods which carry out LiDAR place recognition to determine loop closures. LiDAR SLAM encompasses the techniques necessary to bring these components together to maintain a consistent trajectory and map representation—and to do so in real-time while running on-board a robot, vehicle or sensing system.

Most contemporary LiDAR SLAM systems [66, 843, 282, 265, 1250] are composed of the components shown in the system diagram in Figure 8.7. In the following Section 8.4.1, we will discuss this structure in more detail. We will then focus on the key steps of backend optimization and map update in Section 8.4.2 and describe some common techniques for integrating loop closures to correct the robot trajectory during backend optimization.

Advanced topics for LiDAR SLAM include multi-session and multi-robot mapping. These topics focus on how to fuse multiple mapping sessions into a common global reference frame—which can be either concurrent (running live on multiple robots and solved in real-time) or long-term (aligning multiple maps over time so as to infer environmental change). Multi-session and multi-robot SLAM will be covered in Section 8.4.3, where we will also provide an overview of common solutions and challenges.

Finally, as LiDAR SLAM has matured, it has brought into focus other advanced topics. Safety critical systems such as autonomous vehicles rely on SLAM so we have to consider the robustness and the scalability of LiDAR SLAM system. These topics will be discussed in the final part of this chapter in Section 8.5.

### 8.4.1 Structure of a LiDAR SLAM System

When implementing a LiDAR SLAM system [66, 843, 282, 265, 1250] it is common to decompose it into several modules, one module maintains a relative pose estimate using an odometry estimator, and other modules identify and use loop closures to re-estimate the pose trajectory and to then update the associated map representation to account for this revised pose trajectory. A LiDAR SLAM system usually consists of an odometry estimation module that runs at the sensor frame rate of the LiDAR sensors (*e.g.*, 10 Hz) with the other modules operating at a lower rate (*e.g.*, 1 Hz).

More concretely, consider Figure 8.7. Here, (1) an odometry component estimates a pose in a frame-to-map fashion using the currently active local map. For the odometry module, the most common approach is to register the incoming laser scan to a rolling/active local map (as discussed in Section 8.2). This odometry module will typically only have access to data from the direct vicinity of the sensor —often called an active local map.

Next, (2) a LiDAR-based place recognition method identifies potential loop closure *candidates* as discussed in Section 8.3. Place recognition only identifies that two places (or more specifically observations taken at those two places) are similar. To determine a precise relative transformation estimate between those two places requires fine registration of the corresponding LiDAR scans (typically using ICP).

To initialize the registration, a sufficiently good initial guess of the relative transformation is needed. Geometric priors (taken from the existing pose-graph) can be used for small pose-graphs. Where no geometric prior can be used, modern global registration methods which do not rely on an initial guess but can robustly estimate a relative transformation have been developed [1220, 662]. These methods work well in situations with low overlap between the pair of scans.

Heuristics, such as the travel distance or time difference between two loop closure candidates or the degree of confidence in a RANSAC-based alignment for geometric verification, can be used to determine the validity of a loop closure candidate and to avoid adding false loop closures to the pose-graph.

*pose-graph optimization*

Module (3) is the backend PGO step. It uses the full set of SLAM constraints to solve for an optimized pose-graph and to update the pose trajectory. We will discuss PGO in more detail in the following section.

Finally, in (4) a map update mechanism integrates the sensor measurements into a combined map representation according to this corrected trajectory. There are several potential approaches for this. The most common approach is to use the points directly [256] while approaches such as surfels [282, 843, 66] and implicit representations [1250, 265] attempt to improve the quality of the underlying map or seek to achieve a stronger probabilistic foundation. We refer the reader to Chapter 5 for technical details and discussion about the different dense map representations.

In the next section, we will discuss backend pose-graph optimization in more detail and how the full map representation is typically updated.

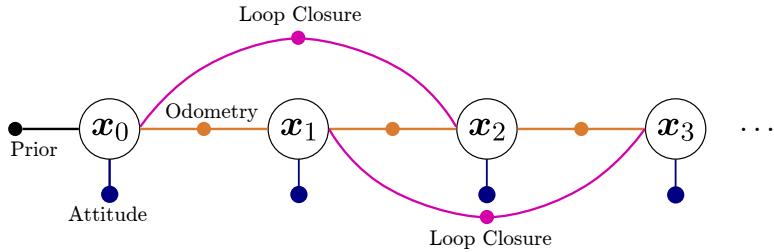


Figure 8.8 A SLAM problem represented as a pose-graph. Each node represents the pose of the sensor whereas the edges represent the constraints coming from odometry (orange) and loop closures (magenta). A Prior Factor fixes the graph origin. Optional Attitude Factors can be used to constrain the pitch and roll when inertial sensing is available. From [884].

#### 8.4.2 Pose-graph Optimization and Map Update

The key part of a LiDAR SLAM system is updating the pose trajectory and map representation after a loop closure has been proposed and verified. As the local pose estimate will contain drift, the error in the local pose estimate needs to be accounted for in an updated pose trajectory. Additionally the existing map representation, integrating the past measurements, will also need to be updated.

Pose-graph optimization (PGO) is typically used in the *SLAM backend*. PGO corrects the estimated trajectory to respect both the odometry constraints and loop closure constraints identified when revisiting already observed places. As introduced and discussed in Part I, see Chapter 1, a factor graph can be used to represent these constraints (as shown in Figure 8.8). Because the graph is made up of only relative pose constraints, it is commonly referred to as a pose-graph. An example of a point cloud map before and after loop closure detection and pose-graph optimization is shown in Figure 8.9.

The constraint set can be optimized using general-purpose solvers such as g2o [612] and GTSAM [253]. To achieve real-time performance, with a pose-graph of increasing size, it is necessary to iteratively resolve a continually growing optimization problem. However, the constraint set is typically sparse—with few interconnected edges. Sparse matrix factorization methods which reorder and relinearize the underlying system of equations allow graphs of over 1000 nodes to be updated in a fraction of a second. For further reading, refer to Chapter 1, iSAM2 [533], and HOG-Man [401].

Note that while 1000 nodes corresponds to a large pose-graph, one must consider scalability. It is common to add odometry constraints only relatively sparsely—not at sensor rate (*e.g.*, 10 Hz) but instead every few meters traveled. Another approach is to subdivide the mapped environment into submaps of fixed physical size (say 30 m traveled) each with a corresponding pose-graph node. It is then assumed that



Figure 8.9 By incorporating loop closure constraints, a SLAM system can create a globally consistent map of revisited locations. The left shows the odometry-only trajectory of a revisited place with visible misalignment between the original visit (blue) and the current visit (purple) to this junction. The right shows the result after pose-graph optimization when loop closures have been integrated resulting in a consistent map of the road junction.

within these submaps the odometry will be locally accurate making re-adjustment of the trajectory inside the submap unnecessary. This approach allows pose-graph SLAM to scale to city-sized maps.

In the backend, pose-graph optimization has to account for pose estimation errors by the odometry as well as errors in the loop closure constraints. To properly model the uncertainty in the pose estimate of the odometry poses, we can also estimate data-driven covariances to distribute the error in the pose-graph optimization sensibly [620, 159] —for example using high covariance of edge constraints where the drift rate is likely to be higher. Finally, to account for incorrect loop closure constraints and bad configurations of the pose-graph, there is a body of research into methods for robust pose-graph optimization [12, 153, 1056, 1057, 1218] which can down-weight, disable or ignore pose-graph constraints which would otherwise cause the map to degrade or diverge (see Chapter 3 for an extensive discussion).

After pose-graph optimization, the full robot trajectory will now be globally consistent, but the effect of this update also needs to be reflected in the map itself. For this purpose, a common approach is simply to re-build the map using the past observations. This would require a system to store the previous observations indefinitely—which can quickly become unsuitable in large-scale environments. An alternative approach is to deform the map representation [843] or to directly link map elements (*i.e.*, such as surfels or submaps) to poses, to allow map deformation in a more scalable manner.

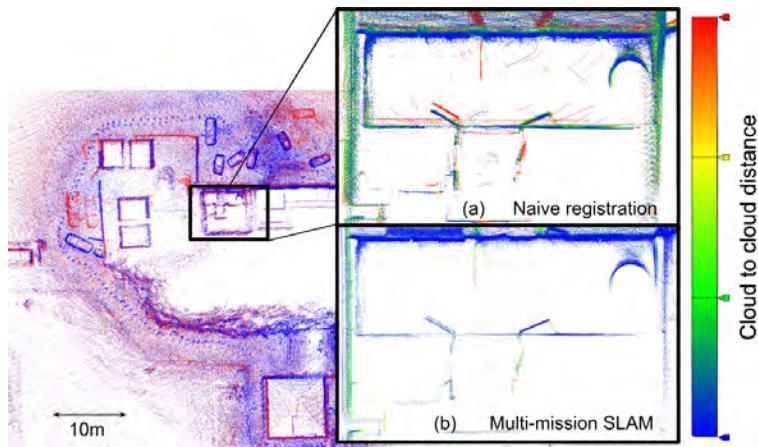


Figure 8.10 Comparison between (a) naive direct alignment of two global point clouds and (b) multi-mission pose-graph relaxation. (a) Point-to-point distances between the two global point clouds shows “double walling” causing phantom change to be hallucinated. (b) Multi-mission relaxation reduces point-to-point distances with structures being more clearly reconstructed. From [948] (©2024 IEEE).

#### 8.4.3 Multi-robot and Multi-session LiDAR SLAM

With the development of mature single-robot single-session LiDAR SLAM systems, there is interest in extending these systems to support multi-robot and multi-session applications. This would be useful because it would allow incomplete maps to be extended into newly scanned territory or for multiple field robots to coordinate their activities using a common map representation. Another use is to co-register maps taken in the same area over time to infer longitudinal environmental change, *e.g.*, for security or monitoring applications.

One initial point which is necessary to make is that while modern LiDAR SLAM systems are increasingly accurate—with **one meter drift per kilometer** being typical in open space—there will always remain some small error within a SLAM map. Simply taking the final point cloud map from two individual SLAM missions and co-registering them will result in locations where point cloud alignment is inconsistent as shown in Figure 8.10.

Initial work in this space focused on how to carry out joint backend optimization of multiple mapping sessions. One approach is to simply transfer the individual constraints from the set of SLAM instances into a single global map representation. An alternative approach is to build each map individually—each with their own coordinate frame, nodes and edges. To link them to one to another, Kim et al. [561] introduced an auxiliary variable called an *anchor node* which accounts for the *anchor* different map coordinates of the individual SLAM missions. This node allows easy

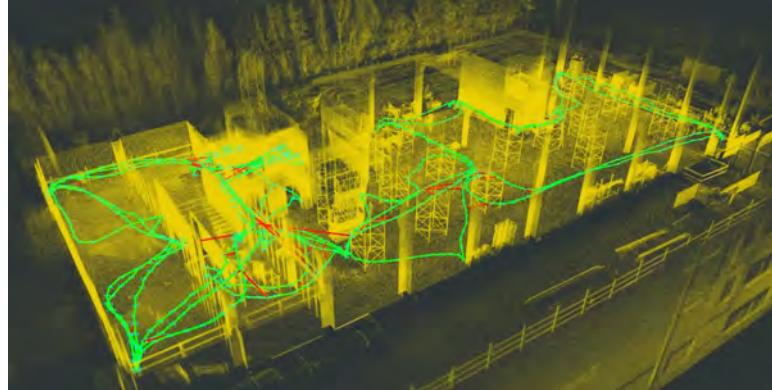


Figure 8.11 A multi-session SLAM map of a construction site. Five different mapping sessions are merged together by establishing inter-session loop closure constraints (in red) and adjusting a joint optimization of the five mapping session trajectories (in green).

global alignment of each individual map and has been used for both multi-session visual SLAM and LiDAR SLAM [746, 564].

As described in [284], aside from the backend optimization, one must determine how loop closure constraints can be established between entirely disconnected SLAM missions. Unlike in the single session SLAM case, there is initially no geometric prior to form the first inter-mission constraint —with multi-session SLAM relying entirely on place recognition to relate maps to one another.

#### *8.4.3.1 Multi-robot SLAM*

Real-time multi-robot SLAM goes one step further, solving the multi-session mapping problem, but doing so with data collected in real-time by robots operating in the field. Estimating a combined map from multiple platforms in real-time allows a robot team to coordinate mission planning, optimally select frontiers of exploration, and to avoid wasted effort returning to a location mapped by another robotic team member. Achieving this capability can allow a team of robots to operate in concert —efficiently exploring territory, identifying which routes are free of obstacles, and perhaps identifying people or objects of interest. This capability is relevant for search and rescue as well as military applications.

One can distinguish between systems which are centralized or decentralized. Centralized systems may transmit sensor measurements to a base station and then compute a combined map at that location. This may be so that the field robot's compute and sensing is kept as simple as possible, for example the mobile robots used by Amazon and Ocado for warehouse operations. On the other hand, decentralized (or distributed) systems would instead build a SLAM map on each individual robot with merging of the set of robot maps being achieved at a base station.



Figure 8.12 Multi-robot SLAM progressed from 2D to full 3D between the 2010 MAGIC challenge to the 2021 DARPA SubT Challenge. The pictures show the winning University of Michigan and Cerberus Teams from the two challenges. Image courtesy of Edwin Olson and Cerberus team.

To describe the evolution of the state of the art we refer to two major international multi-robot exploration challenges. The first one is the Multi Autonomous Ground-robotic International Challenge (MAGIC), which was held in Brisbane, Australia in 2010. This challenge involved teams of wheeled robots executing a reconnaissance mission in a  $500\text{ m} \times 500\text{ m}$  challenge area to correctly locate and classify simulated threats. The winning team, Team Michigan, fielded 14 3D-printed robots equipped with 2D LiDAR scanners [825] as well as cameras (to identify loop closures). Each robot carried out 2D LiDAR odometry and transmitted its pose-graph constraints to a base station which assembled a global 2D multi-robot map.

Research progress over the last decade was evidenced by the DARPA Subterranean Challenge [599] which was held in Louisville, Kentucky in 2021. It posed a similar challenge to competing teams at MAGIC—to explore unknown environments—but with the robots operating in more complex 3D underground environments with stairs, curbs, and ramps. Here 3D multi-beam LiDAR was heavily used but also augmented with visual odometry, wheel/legged and, in some cases, thermal odometry to overcome degenerate circumstances where LiDAR odometry can fail such as in narrow environments in the underground tunnels.

The SubT teams published an overview article which provides a comparison between the fielded systems [299]. Each team used a semi-decentralized approach with individual robots building pose-graph-based SLAM maps on board with a multi-robot SLAM map created at a single central base station. Key challenges included compression and communication as the robot teams needed to maintain a dynamic wireless mesh network to transmit data back to this base station. For example, the WildCat SLAM system from CSIRO [600] was notable for using a compressed surfel representation to represent local submaps. These surfel maps took up much less space than the raw point clouds—greatly reducing the bandwidth needed to

transmit the map and pose-graph constraints to the base station computer. During the finals, a complete map took only 21.5 MB per robot.

As mentioned above, the most complex problem is fully distributed SLAM system—where each robot platform is tasked with building and maintaining a representation of the overall combined map subject to communication and scaling constraints. Some existing approaches [493, 1092] have explored how to do this and focused on the mechanisms to share the set of constraints and local submaps progressively with each robot. Issues of consistency are key in this topic.

### 8.5 Further Readings & Recent Trends

LiDAR SLAM has seen significant advancements over the last decades—especially since the introduction of LOAM [1266]. Improved odometry with high accuracy [1208, 996, 1130, 256] and efficient pose-graph SLAM systems [66, 265, 843, 900] have also been developed. However, despite this progress, there are still unsolved problems and challenges to tackle.

**Robust and Resilient Perception.** A recent robustness evaluation by Zhao et al. [1285] identified that LiDAR SLAM systems struggle to perform effectively in cluttered and unstructured environments. Structure-less corridors, underground mines and extreme weather conditions such as snow, fog, and dust are other challenging situations pointed out in a review by the DARPA Subterranean Challenge competitors [299].

Furthermore, the performance of current LiDAR SLAM systems is typically demonstrated experimentally and lack formal robustness evaluation. Best performance is achieved through feature engineering and manual parameter tuning, which perhaps ought to be dynamically adjusted according to the operational scenario as in KISS-ICP [1130]. Future improvements in this regard should consider actively adapting algorithm behavior to account for changes in the environment through introspection.

With regard to place recognition, there are a broad spectrum of research directions. Key survey papers such as [1004, 1246] offer a comprehensive foundation on the topic. Ongoing research includes methods for robust retrieval which generalize across the various categories of LiDAR sensors [528]. Other research looks to achieve heterogeneous place recognition between LiDAR and other modalities such as radar [1245] and OpenStreetMap [211]. Researchers have also successfully leveraged the LiDAR’s intensity information [997, 1148], alongside traditional XYZ data to improve performance. Long-term place recognition across multiple mapping sessions [565] is another promising research topic; as is change detection and lifelong map management [564, 1248].

**Multi-Sensor Fusion.** Fusing multiple sensors with complementary characteristics is a key route to more robust and resilient robotic systems. Degraded perception of a particular sensor can be ameliorated by fusing a different and complementary

sensor, *e.g.*, radar works well in rain or smoke; or using visual feature tracking in a tunnel where LiDAR fails [1188, 1284]. However, when integrating additional sensors with LiDAR, we inevitably acquire a plethora of sensor data, leading to redundancy. There are open questions about how to achieve a balance between redundancy and lightweight computation. Furthermore, one must consider how to efficiently select the most reliable information when fusing estimates of multiple sensors. Solutions range from early fusion approaches (using a single tightly coupled estimator) and late fusion approaches (where separate individual-sensor pose estimators are combined).

Finally, another practical consideration is that multi-sensor systems may lack accurate calibration and individual sensors may not be precisely synchronized. This places a burden on the underlying estimation system making full, tight sensor fusion difficult to practically use over extended time intervals. There is still space for research into these intriguing research questions.

**Uncertainty and Bayesian Estimation.** Closely related to the question of how sensors can be reliably fused is the question of uncertainty estimation in LiDAR SLAM. Properly calibrated measures of sensor uncertainty are required to probabilistically fuse multiple pose estimates. However, most successful LiDAR-based approaches rely on ICP, which does provide a calibrated and robust estimate of the pose uncertainty.

While some early approaches [620, 159] for approximating covariance exist, the estimated uncertainty used in LiDAR SLAM system is often unreliable. As a result, algorithms often use fixed odometry covariances during backend pose-graph optimization. A more introspective handling of uncertainties in the odometry process has the potential to address degraded pose estimates at an earlier stage. There are still many open questions regarding uncertainty estimation, where robust solutions would support the development of more resilient SLAM systems as well as multi-modal SLAM.

**Deployment in Closed-Loop Autonomous Systems.** A final consideration is how LiDAR SLAM performs in a desired final application. These applications are as varied as light-weight aerial vehicles flying through forests, handheld devices scanning construction sites, and self-driving cars operating in poor weather. The traditional electronics requirements of Size, Weight, and Power (SWaP) are augmented with additional parameters of accuracy, robustness, computation, and latency. For instance, the ability for a self-driving car to respond to a potential upcoming collision with as little delay as possible is affected by the computational latency of the LiDAR system. Thus in certain applications the most accurate and computationally complex system may not always be the preferred solution.

# 9

## Radar SLAM

Martin Magnusson, Christoffer Heckman, Henrik Andreasson, Ayoung Kim,  
Timothy Barfoot, Michael Kaess, and Paul Newman

*radar*

In this chapter, we explore the use of radar (RAdio Detection and Ranging) in SLAM. Compared to cameras and lidar, radar is somewhat undersubscribed. However, due to its ability to work in poor visibility, at long range, and to natively produce velocity information, its popularity is on the rise. We begin by discussing the types of radar sensor typically used in robotics, their unique sensing principles, and some of the challenges that come along with radar (Chapter 9.1). We then discuss radar filtering, radar odometry (Chapter 9.2), place recognition (Chapter 9.3), and finally SLAM (Chapter 9.4); Figure 9.1 shows how these pieces fit together. We conclude with a discussion of radar SLAM datasets (Chapter 9.5) and an outlook on the use of radar moving forward (Chapter 9.6).

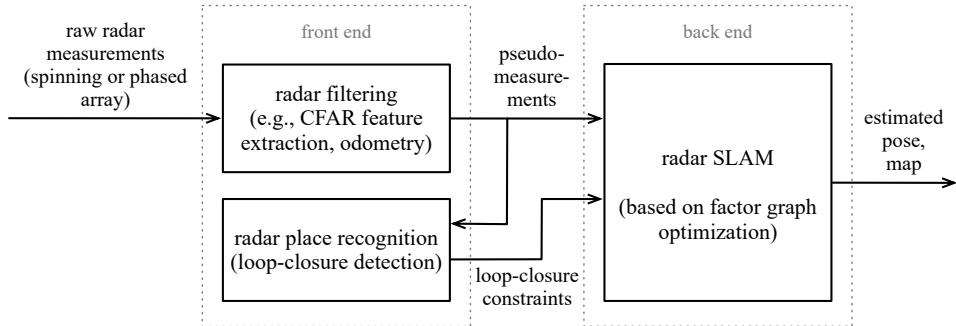


Figure 9.1 The flow of information in radar-based SLAM follows the same pattern as other SLAM systems with the details of each process being slightly altered to suit the specifics of radar.

### 9.1 Introduction to Radar

#### 9.1.1 Sensor Types

In the following section, we introduce two of the most common radar types that are encountered in robotics: *spinning radar* and *system-on-a-chip (SoC) radar* (see

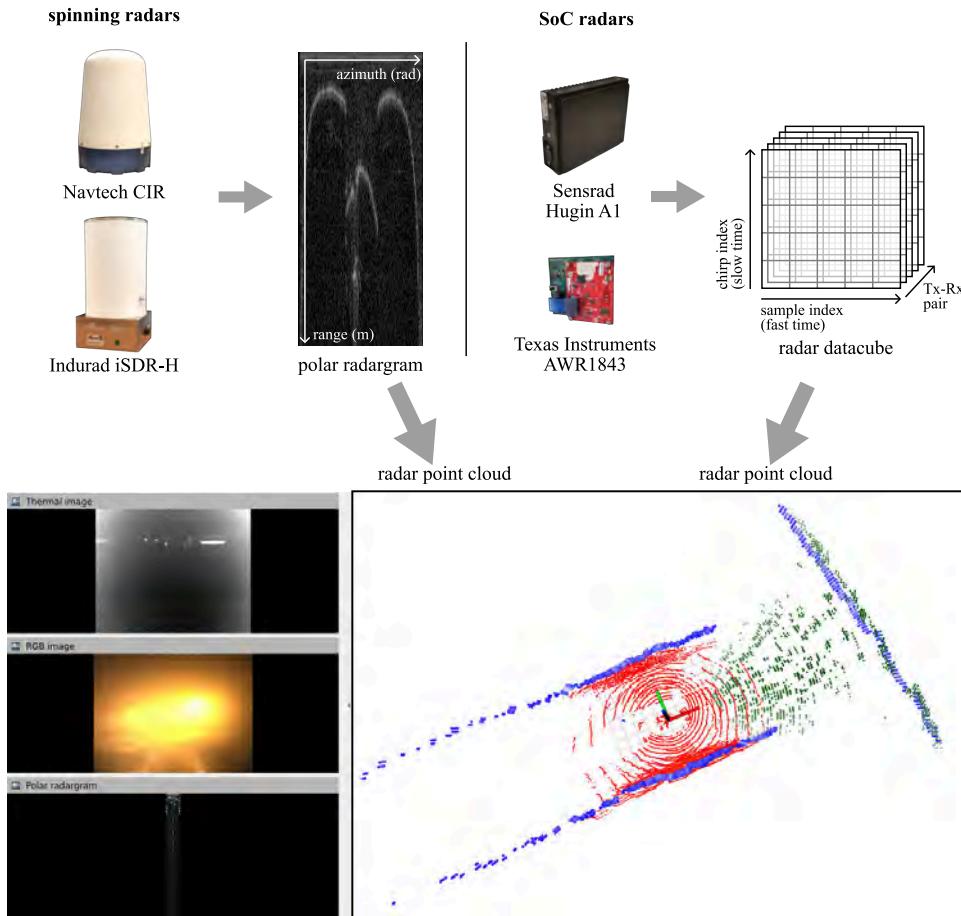


Figure 9.2 The two main categories of radars used in robotics are *spinning* (top left) and phased-array *SoC* (top right). Some examples of each are shown along with the main data product produced by each type of sensor. Often these raw data products are subsequently turned into a sparse point cloud using radar filtering (bottom). The bottom right part shows three point clouds from a tunnel with smoke. Red: LiDAR point cloud with much limited range due to the smoke. Blue: point cloud from 2D spinning radar (tunnel walls in all directions are clearly visible also at a distance). Green: 3D point cloud from SoC radar (walls and ground surface in front of the vehicle are visible). The bottom left part shows thermal and visual/RGB image data from the same scene, and the polar radargram from which the blue point cloud is extracted.

Figure 9.2). Each has its strengths and drawbacks. They differ mainly in how they are ‘actuated’ with spinning radar mechanically rotating a single antenna and SoC using multiple antennas whose signals are combined to deduce the angles and ranges of objects reflecting the transmitted signals.

*spinning radar*  
*radargram*

Utilizing a rotating radar sensor, a *spinning radar*—sometimes referred to as scanning radar or imaging radar—crafts precise polar representations of its surroundings, exemplified in Figure 9.2. The main data product produced is a *polar radargram*. Imaging radars are distinguished by their ability to detect objects at distances exceeding 100 meters. In some modes, the velocity of those objects can also be determined by exploiting the Doppler effect.

In contrast to LiDAR systems, spinning radars are limited to providing data on a two-dimensional plane, lacking the capacity to measure the elevation of detected objects. This limitation persists even though reflections might originate from various elevations within the antenna’s vertical beamwidth. Additionally, because their operational mechanism involves transmitting and receiving a single pulse for each antenna angle, these radars sometimes do not offer velocity data, which requires multiple pulses for calculation. More recent work uses the signal from multiple neighbouring azimuths to recover velocity.

Constructing a mechanically spinning 3D radar with multiple vertical beams, similar to currently widespread 3D LiDAR sensors, is not practically feasible due to the much larger size of the antennas and focusing mechanisms compared to laser diodes.

*SoC radar*  
*radar datacube*

*System-on-a-chip (SoC) radars* integrate processing units within a minimal set of chips, which are either directly mounted on patch antennas (arrays of transmitters / receivers) or incorporated into the printed circuit board itself. SoC radars are characterized by their lightweight design and reduced power requirements compared to spinning radars, thanks to their integrated architecture and lack of moving parts. The performance in terms of accuracy and resolution for SoC radars hinges on the design of the antenna array and the proprietary processing techniques manufacturers employ to integrate measurements from multiple antennas. The primary output produced is a *radar datacube*.

SoC returns are typically mapped in spherical coordinates by azimuth, elevation, and range. Given that radial velocity adds another dimension, these radars are often referred to as 3+1D or 4D. Conversely, systems with limited or absent vertical resolution, due to a scarcity of vertically aligned antennas, are denoted as 2D array radar systems, producing 2+1D data products.

### 9.1.2 Radar Sensing Principles

In this section, we will discuss basic operations, antennas and datatypes, and challenges as they relate to robotics applications. We outline each in the following

sections and designate how radar differentiates itself from other rangefinding sensors.

MmWave (millimeter-wave) radar is designated by radar systems whose electromagnetic wavelengths are between 1–10 mm with frequency ranging from 30–300 GHz. Within this range, most radar sensors operate in the 76–81 GHz segment of the spectrum due to automotive applications such as ADAS systems having spectrum carve-outs in this range.

*mmWave*

#### 9.1.2.1 Radar Cross Section

The process of mmWave radar involves emitting electromagnetic pulses that travel until they meet objects, bouncing back towards the radar. The Radar Cross Section (*RCS*)<sup>1</sup>, influenced by an object's material composition, size, and shape, plays a crucial role in determining how strongly each object reflects these electromagnetic pulses. Essentially, the RCS represents the size of a hypothetical sphere that would reflect the same amount of energy as the target object. Thus, larger and more solid structures such as vehicles and thick concrete walls exhibit a higher RCS compared to smaller objects or pedestrians, which present a lower RCS.

The term *radar intensity* refers to the strength of the radar's echo from an object, which is a function of the radar's transmitted power and the object's RCS. In essence, this intensity is chiefly determined by the radar's emitted power and the RCS of the encountered target. Literature highlights that this measure of intensity has been crucial for extracting semantic details about objects or aiding in navigation, as stronger echoes tend to be associated with distinctive and easily recognizable features in the environment [1237]. The strength of the radar signal is influenced by several additional factors, including the type of antenna used, the characteristics of the electromagnetic pulse emitted, and the antenna's ability to detect returns from objects.

#### 9.1.2.2 FMCW Ranging

A radar comprises at least one transmitting antenna (TX) and one receiving antenna (RX).<sup>1</sup> In the context of FMCW radar, the TX antenna's role is to emit an RF pulse that steadily increases in frequency, known as a *chirp*. This is often called *sawtooth modulation*; we will also discuss *triangular modulation*, another chirp-based technique, later on. These chirps bounce off objects in the environment, and the RX antenna captures the echoes (see Figure 9.3). Upon receiving a chirp, the signal is amplified and then mixed with, or subtracted from, the original TX chirp to generate an Intermediate Frequency (IF), which is also a signal. The mixing process results in an IF that is a sine wave of constant frequency  $f_0$ , due to the identical slope of both the transmitted and received signals. The performance of the radar system, including its range and velocity detection capabilities, is affected by

*FMCW*  
*chirp*

<sup>1</sup> Although the TX and RX antennas can occasionally be the same unit, most SoC sensors opt to separate them, allowing for the incorporation of multiple TX and RX antennas.

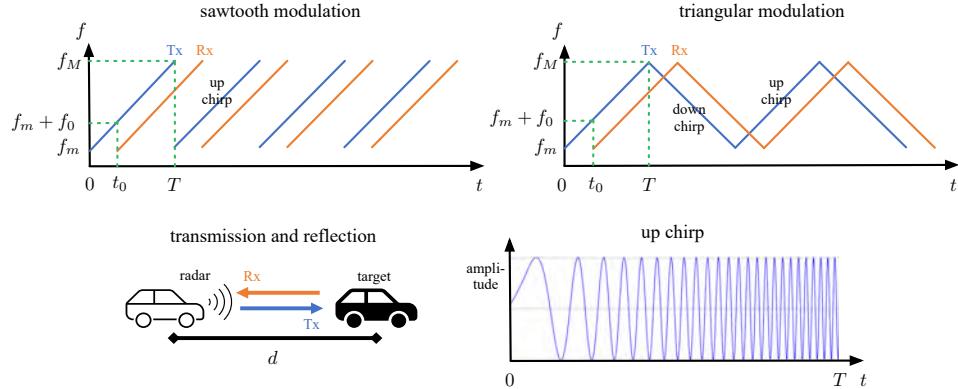


Figure 9.3 A frequency modulated continuous-wave (FMCW) radar works by emitting chirps (waves of increasing or decreasing frequency) that are reflected off targets and then received. Range (and velocity) are determined by analyzing the frequency (and phase) shifts of the reflected signal compared to the transmitted one. Two common frequency modulation strategies are *sawtooth* and *triangular*; the advantage of the latter being that the Doppler frequency shift can be disentangled from the range shift.

various chirp parameters such as the bandwidth (the difference between the initial and final frequencies), the chirp slope, and the duration between chirps.

The main idea of this *frequency modulation* is to encode temporal information onto a continuous wave, enabling the execution of range calculations. This technique stands in contrast to *amplitude modulation*, where the precision of the returned signal depends exclusively on the frequency's bandwidth. This characteristic renders frequency modulation more resilient to issues such as signal-to-noise ratios and the RCS of targets, which might otherwise affect the clarity of individual returns. In the context of FMCW radar, where waves are interspersed with unique intervals between pulses in a train, the signals maintain coherence. This coherence allows for each chirp to be accurately associated with its originating transmit-receive pair and its position within the sequence, facilitating the correlation of signals in terms of both amplitude and phase. This represents a significant evolution from the older time-of-flight pulsed radar systems, which relied on incoherent amplitude measurements and required the expertise of skilled operators to sift through clutter and isolate significant signals.

In FMCW, the calculation of distance relies on the temporal gap between when a signal is sent and when its echo is received. Utilizing the speed of light,  $c$ , and the initial arrival time,  $t_0$ , the distance,  $d$ , to an object is computed as

$$d = \frac{c}{2} t_0. \quad (9.1)$$

The top-left portion of Figure 9.3 shows the region where an IF signal is generated

using green dashed lines. The difference between the transmitted and received signal is an IF signal. The IF signal is a sine wave  $\sin(2\pi f_0 t + \phi_0)$  whose frequency is proportional to a constant  $f_0$  that spans from  $t_0$  to  $T$  that only depends on the distance to the target, and is offset in phase by  $\phi_0$ .

The frequency,  $f_0$ , is defined as a function of the distance to the target, the duration of the transmitted chirp,  $T$ , and the bandwidth of the transmitted signal,  $B = f_M - f_m$ . The bandwidth and transmit time are related to the slope of the chirp,  $S = B/T$ , as

$$f_0 = \frac{2Bd}{Tc} = \frac{2Sd}{c} \Rightarrow d = \frac{cf_0}{2S}, \quad (9.2)$$

where we now have the distance,  $d$ , as a function of the (measured) intermediate frequency,  $f_0$ .

In this section, although the equations are presented within the frequency domain, the real-world capture of signals predominantly occurs through digital sampling. This sampling is done at a high rate, typically every 100 nanoseconds or less, using a high-frequency Analog-to-Digital Converter (ADC). Following this, the sampled data undergoes a sequence of Fast Fourier Transform (FFT) operations. These operations generate graphs depicting frequency and amplitude, from which signal peaks can be discerned. The identification of range frequencies is achieved by applying FFT to the data from a single chirp and its corresponding return signal. To calculate velocity frequencies, a series of FFTs are executed on multiple chirp and return sequences.

#### 9.1.2.3 Determining Distance and Velocity with Sawtooth Modulation

Sawtooth frequency modulation (see Figure 9.3) is typically used with SoC radars and some spinning radars. The phase of the IF signal,  $\phi_0$ , can be expressed as a function of the wavelength  $\lambda$  of the signal and the distance  $d$ :

$$\phi_0 = 2\pi f_m t_0 = \frac{4\pi d}{\lambda}. \quad (9.3)$$

While both the base frequency  $f_0$  and phase  $\phi_0$  are functions of distance  $d$ , the phase is only valid for sufficiently small distance values and is subject to angle-wrapping. Thus this is typically not used for range estimation but to measure small changes  $\Delta d$  where the phase responds linearly in velocity estimation.

For radial velocity estimation (see Figure 9.4), at least two sequential up chirps are employed as shown in the top left of Figure 9.3. As the distance in time between each chirp in the sequence is small, on the order of 40 microseconds, the range measurement from both samples and thus the relative IF  $f_i$  and  $f_{i+1}$  are nearly identical. However, the IF signals will possess distinct phases. This phase disparity  $\Delta\phi$  corresponds to a motion of the object. The estimated velocity  $v$  is determined

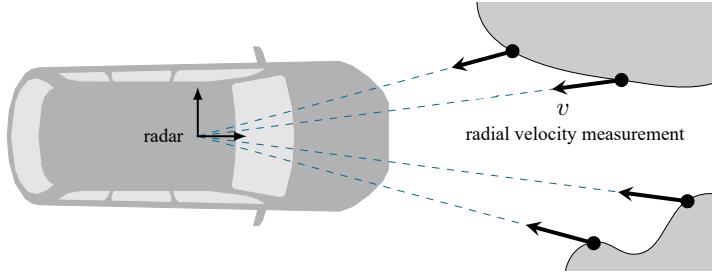


Figure 9.4 Radar is able to use the Doppler effect to measure the *radial* velocity between a unit and the scene it is imaging.

from the phase difference,

$$\Delta\phi = \frac{4\pi\Delta d}{\lambda} = \frac{4\pi vT}{\lambda}, \quad (9.4)$$

simplified to

$$v = \frac{\lambda\Delta\phi}{4\pi T}. \quad (9.5)$$

As velocity is a function of phase, we note the maximum detectable velocity is unambiguous for  $|\Delta\phi| < \pi$ . Thus  $v_{\max} = \lambda/(4T)$  is a function of the wavelength of the signal and the time between chirps.

#### 9.1.2.4 Determining Distance and Velocity with Triangular Modulation

Triangular frequency modulation (see Figure 9.3) can also be used; for example, it is sometimes used with spinning radars. In reality, when an intermediate frequency is derived from an up chirp, it comprises two components: (i) frequency shift due to the time of flight of the signal (discussed already),  $f_{0,t}$ , and (ii) the apparent frequency shift due to the *Doppler effect* if there is a relative velocity between the radar and the target,  $f_{0,d}$ :

$$f_{0,\text{up}} = f_{0,t} + f_{0,d}. \quad (9.6)$$

However, if we follow an up chirp with a down chirp that reflects off the same target, the sign of the temporal frequency shift will flip while that of the Doppler shift will not:

$$f_{0,\text{down}} = -f_{0,t} + f_{0,d}. \quad (9.7)$$

From these two equations we can solve for  $f_{0,t}$  and  $f_{0,d}$ :

$$f_{0,t} = \frac{f_{0,\text{up}} - f_{0,\text{down}}}{2}, \quad f_{0,d} = \frac{f_{0,\text{up}} + f_{0,\text{down}}}{2}. \quad (9.8)$$

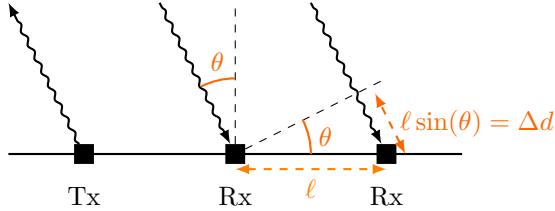


Figure 9.5 Angle of arrival estimation with phased-array radar. The measured difference in phase of the signal emitted by the TX transmitter antenna as received by two RX receiver antennas at a distance  $\ell$  corresponds to the angle  $\theta$  to the target.

Finally, from these two components, we can calculate the range and velocity according to

$$d = \frac{cf_{0,t}}{2S}, \quad v = \frac{cf_{0,d}}{2ST}. \quad (9.9)$$

Notably, the range calculation presented earlier in (9.2) is not corrected for the Doppler effect whereas this one is. The downside of using triangular modulation is an increase in latency since we now require slightly older data in the calculation of range and velocity. However, we do not need to work with the phase of the signal.

#### 9.1.2.5 Determining Angle for SoC radar

For spinning radar, determining the angle to a target is trivial since the beam is focused in a single azimuth direction at each time. Angle estimation for SoC radar is slightly more complex but can be achieved with a similar phase difference calculation as above.

Given multiple receiver units separated by an interval  $\ell$ , the distance disparity  $\Delta d$  emerges in reflections. The angle of arrival  $\theta$  can be derived from the modification of (9.3) with the geometric relation  $\Delta d = \ell \sin \theta$ . The received signal must travel an extra distance  $\ell \sin \theta$  to reach the second receiver antenna, as illustrated in Figure 9.5. This corresponds to a phase difference of  $\Delta\phi = (2\pi/\lambda)\ell \sin(\theta)$  between the signals received at the two RX antennas. Given the measured phase difference  $\Delta\phi$ , the angle of arrival  $\theta$  can be computed as

$$\theta = \arcsin \frac{\lambda \Delta\phi}{2\pi\ell}. \quad (9.10)$$

While one TX and two RX antennas are sufficient in principle for determining the angle to a target, having more than two RX antennas enables higher resolution and thus the ability to distinguish multiple nearby targets. The phase of the returned signal will be offset by an additional  $\Delta\phi$  at each RX. Sampling the signal across the RX antennas and performing an FFT on this signal sequence can be used to reliably estimate  $\Delta\phi$ .

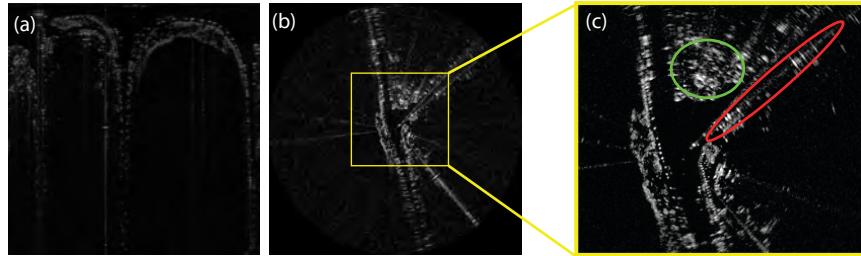


Figure 9.6 A polar radargram in (a) transformed to Euclidean coordinates in (b), where we can observe several types of radar noise that are unique compared to other sensors from a zoomed view in (c). Speckle noise returns are the most common, with ambiguous clutter circled in green. Multipath reflections develop where returns bounce off nearby walls or the ground before hitting the antenna, generating reflections of true targets. A series of repeated returns is circled in red. The original image is sampled from the Mulran dataset [566].

MIMO

The above example illustrates a “SIMO” phased-array radar system (single input, multiple output). Most radar sensors used for SLAM applications are MIMO (multiple input, multiple output) with several TX and RX antennas. Rather than doubling the number of RX antennas, it is possible to achieve the same resolution by adding one more TX antenna, as long as the RX antennas can distinguish the signals from the multiple TX antennas. Different techniques can be used to ensure that the TX signals are uncorrelated (orthogonal); *e.g.*, frequency division (where each transmitter uses a different frequency band), code division (where each transmitter sends a signal modulated by a unique code sequence), or time-division multiple access (TDMA) where each transmitter uses a different time slot. The same principle can also be applied to 2D TX-RX arrays that can measure both azimuth and elevation angles, thus producing 3+1D data.

### 9.1.3 Challenges to Radar Applications

Radar technology, like any sensor system, presents a range of challenges that necessitate careful consideration during development. These challenges include a variety of noise types that are particularly prominent in radar, such as multipath reflections, biased and sparse range readings due to the wide beam width, receiver saturation, and speckle noise. Illustrations of some of these noise phenomena are provided in Figure 9.6, which depicts a polar radargram transformed to Euclidean coordinates. We discuss some relevant radar filter techniques in Section 9.1.4.

#### 9.1.3.1 Speckle Noise

Noise in radar measurements come from several sources, including thermal noise, electronic flaws, and varying RCS of targets. When a radar emits an electromagnetic

pulse, it captures the energy reflected back by all objects within the antenna's field of view. The interaction of this pulse with objects scatters the radar waves, leading to constructive and destructive interference. Such interactions can either produce false signals or cancel out legitimate returns received by the antenna, irrespective of the signal's origin. These factors contribute to signal variations across the frequency domain, where the most prominent peaks represent a mix of genuine targets and false alarms. In the absence of a mechanism to distinguish between genuine and false returns, the sensor ends up generating a pattern of scattered points, commonly referred to as speckle noise. For accurate identification of landmarks, crucial for pose estimation and feature matching, it becomes essential to estimate the uncertainty around these reflections, possibly over several scans.

#### 9.1.3.2 Multipath

Beyond speckle noise, multipath returns constitute another form of measurement errors, originating from varied detection paths associated with a single object. Imagine a scenario with a landmark situated in front of the sensor. While some transmitted rays may directly reach this landmark, others might only arrive at the antenna after reflecting off the ground or bouncing off a wall. To the radar, it appears as though the landmark is located beneath the road or beyond the wall, leading to the perception of what are termed 'ghost objects' or static outliers. The elimination of these outliers is crucial for ensuring the reliability of point cloud mapping or localization.

#### 9.1.3.3 Motion-Induced Distortion

Scanning sensors, including both lidar and radar, inherently exhibit motion distortion. This is particularly true for spinning radar, which constructs each polar image through a single rotation. Consequently, if the sensor is moving, the position of a single object captured at the start and end of one rotation will differ. This discrepancy becomes significant with sensors operating at low frequencies or when the vehicle moves swiftly. For instance, the Navtech CIR 304, a commonly used imaging radar, operates at a frequency of 4 Hz, posing challenges for accurately aligning raw frames. There are also faster spinning radars like the newer Navtech RAS3 which spins at 10 Hz, similar to many lidars, and the Indurad iSDR that can spin at up to 50 Hz. Still, when the sensor is mounted on a fast-moving platform (like a car), the motion-induced distortion can be significant.

#### 9.1.4 Radar Filtering

The occurrence and distribution of false targets (Section 9.1.3) can change over time and are characterized by unpredictable parameters, rendering static filtering approaches such as simple thresholding insufficient, as they may allow false alarms to pass through. In response, researchers have devised methods to dynamically

estimate the distribution of false alarms, taking these challenges into account. These filtering methods typically consider the measurements along one azimuth direction, trying to estimate which range bin(s) contain true targets and which are false alarms.

#### *CFAR*

The constant false alarm rate (CFAR) filter [808] is a popularly implemented method designed to sustain a specified probability of false alarms amidst dynamically changing and uneven interference. The process begins by segmenting a signal—such as the frequency domain representations obtained post-FFT of radar ADC samples—into discrete segments known as cells. These cells are then assessed using a sliding-window approach. At the core of this window lies the set of cells under test (CUT). The intensity of the CUT is evaluated against that of the adjacent cells, referred to as training cells, which precede and follow the CUT. In some implementations, guard cells may be placed between the training cells and the CUT to prevent the local influence of the CUT from affecting the training cells' magnitude. A decision to accept or reject a CUT set is made based on whether its intensity surpasses a calculated threshold, which is derived from the comparative intensity of the surrounding training cells, guard cells excluded. Several variants of CFAR exist, the canonical version being cell-averaging CFAR (CA-CFAR), where the threshold is computed in relation to the mean power of the training cells, scaled by a threshold multiplier that is selected from the desired probability of false alarm. A common alternative is ordered-statistic CFAR (OS-CFAR) where a more robust statistic such as the median of the training cells is used instead of the mean.

Figure 9.7 illustrates how the CA-CFAR threshold adapts to a radar signal and which range bins are selected as targets vs noise, compared to two other filtering methods.

A variant of CFAR designed and tested specifically for radar odometry is BFAR (bounded false alarm rate) [25] which simply modifies the output  $Z$  computed from a CFAR detector with an affine transformation  $T = aZ + b$ , where  $b$  is a learnable parameter that scales the output to blend between the CFAR output and a fixed-level threshold.

While CFAR and its variants are used in many radar applications, several pipelines for radar odometry and SLAM employ simpler filtering strategies. Whereas CFAR was developed for target detection (where it may be important not to miss a weak detection), when filtering radar for use in odometry the concern is rather to retain only those points that can reliably be detected over time and from different viewpoints. One popular technique involves selecting the  $k$  strongest returns above a static threshold along each azimuth, with  $k$  ranging from 1 and upwards. A statistical threshold may also be employed, selecting all points with an intensity higher than one standard deviation over the mean value.

Some recent approaches instead use machine learning techniques to increase the accuracy and resolution of radar output, typically using LiDAR data as ground truth for training. Cheng et al. [195] use a generative adversarial network (GAN)

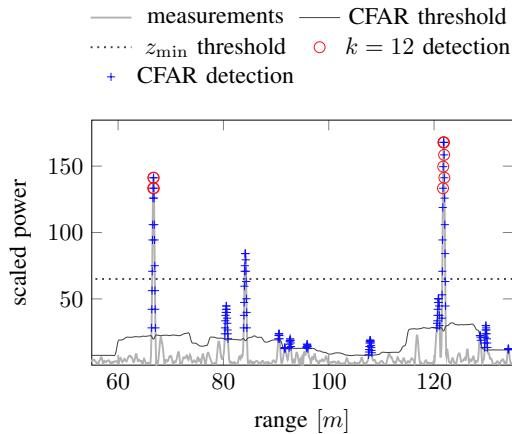


Figure 9.7 Examples of radar filtering, comparing a CFAR filter with a constant power threshold and a  $k$ -strongest strategy. The power/range plot along one azimuth direction is plotted in grey. Returns from true targets appear as spikes in the plot but there are also several ambiguous peaks. CFAR produces an adaptive threshold, plotted with a black line. Detections reported by the CFAR filter are plotted in blue, and in this case includes several “false alarms”. The  $k$  strongest filter (with  $k = 12$  in this case) is more conservative and only returns points around the main targets. Figure from Adolfsson et al. [6] (©2021 IEEE).

to generate point clouds based on range-Doppler velocity matrices. Xu et al. [1207] train a regressor and classifier, where the regressor outputs improved, higher-resolution depth readings, and the classifier provides an estimate of whether the data is out of range. These methods strive to learn models that can retain only those returns in the radargram that correspond to a surface that would be detected by a LiDAR.

## 9.2 Radar Odometry

The goal of radar odometry is to, given a set of ordered radar readings over time, estimate the egomotion of the sensor. A radar odometry approach typically involves handling an intermediate representation, such as a set of the last  $N$  radar readings or a continuously pruned local map. The focus lies on obtaining an accurate pose estimate at a local scale. Without considering explicit loop closures, the error will eventually accumulate without bounds even for good odometry methods. Methods using spinning radar may accumulate on the order of 1–2 % translational drift per 100 m.

A particular feature of many radar sensors is the per-point ‘Doppler’ velocity estimates, which can be used to estimate odometry in a correspondence-free manner, as described in Section 9.2.1. In addition to Doppler-based methods, the relative



Figure 9.8 Example of open-loop radar odometry on the Oxford Radar Robotcar dataset [127], using the CFEAR method [8] with data from a Navtech 2D spinning radar. The ground-truth trajectory plotted in blue and the odometry estimate in orange. Point targets extracted from the radargram in grey.

transformation between two nearby radar scans is often estimated via spatial correspondences so as to determine which parts of one scan can be found in the other scan. Given a set of such correspondences, a distance metric can be computed and optimized. Depending on the type of scanner, how to obtain these correspondences is different; a spinning radar often produces a raw signal that either can be used directly as described in Section 9.2.2, to extract higher-level features containing information from the raw signal (see Section 9.2.3) or to extract range points that can be used in registration, much like in LiDAR odometry (see Section 9.2.4).

An indicative example of what open-loop radar odometry using a 2D scanning radar may look like is shown in Figure 9.8.

### **9.2.1 Doppler Odometry**

The radial velocity obtained from Doppler measurements can be used to directly estimate the sensor’s linear velocity. In Doer and Trommer [272], Doppler information is utilized via a combination of three-point RANSAC and a least squares problem to estimate linear velocities.

However, the rotational velocity component is not directly observable from the per-point velocity measurements in the data from a single radar —unless assumptions can be made about the kinematic model for the radar system, for example,

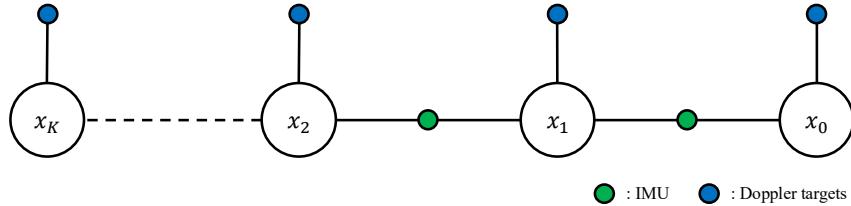


Figure 9.9 Factor graph representation of the radar-inertial velocity estimation system. States from  $K$  previous timesteps are jointly estimated using Doppler targets and sets of IMU measurements as constraints. Figure adapted from Kramer et al. [604].

knowing where the radar is mounted with respect to the center of rotation and assuming no skidding [550, 354]. Therefore it is common to use IMU data (or more specifically, a gyroscope) for radar odometry systems that rely solely on Doppler information from a single radar. In Huang et al. [490], a consumer grade IMU combined with cascaded SoC radars achieves low drift in diverse 3D indoor spaces. Kubelka et al. [607] compared several variants of registration-based approaches for 3+1D radar with registration-free Doppler + IMU odometry [272] and found the registration-free method to produce the lowest error, not least in feature-sparse environments, with a drift as low as 0.3% over a 4.5 km trajectory reported.

Kellner et al. [550] show how linear and rotational velocity can be estimated from 2+1D radar data when the radar is mounted on a vehicle with Ackermann steering and the mounting point of the radar sensor with respect to the center of rotation of the vehicle is known. Galeote-Luque et al. [354] extend this to the 3+1D case and estimate five degrees of freedom (linear motion in three dimension plus yaw and pitch rotation, but not roll).

Since the Doppler-based modality of odometry is rather specific to the radar methodology, we provide an example based on Kramer et al. [604].

#### *Example: Doppler Odometry Factor Formulation*

A straightforward approach to integrating a Doppler factor from radar is in estimating the body-frame velocity of the sensor platform over a sliding window of  $K$  previous radar measurements. These velocities are interconnected through integrated accelerometer measurements from the IMU, which can form a comprehensive system for accurate velocity estimation. The system's structure can be represented using a factor graph, where states from  $N$  previous time steps are jointly estimated using Doppler targets and sets of IMU measurements as constraints.

IMU measurements (accelerometers and gyros) are typically affected by both bias  $\mathbf{b}$  and gravity  $\mathbf{g}^w$ . Since velocity estimates derived from radar data are free from bias, accelerometer biases can be compensated for by including them in the state vector. However, compensating for the effects of gravity requires estimating

the IMU's attitude, specifically its pitch and roll, which are represented by the orientation quaternion  $\mathbf{q}_s^w$ . Consequently, the full state vector is expressed as  $\mathbf{x} = \begin{bmatrix} \mathbf{v}^{s\top} & \mathbf{q}_s^{w\top} & \mathbf{b}^\top \end{bmatrix}^\top$ .

The radar-inertial ego-velocity estimation is formulated as an optimization problem, where the cost function integrates the constraints and measurements to provide an accurate estimate of the sensor platform's velocity:

$$J(\mathbf{x}) = \underbrace{\sum_{k=1}^K \sum_{d \in \mathcal{D}_k} w_{d,k} e_{d,k}}_{\text{Doppler term}} + \underbrace{\sum_{k=1}^{K-1} \|\mathbf{e}_k\|_{\mathbf{W}_k}^2}_{\text{inertial term}}, \quad (9.11)$$

where  $K$  is the number of past radar measurements for which states are estimated,  $\mathcal{D}_k$  is the set of targets returned from the radar measurement at time  $k$ ,  $e_{d,k}$  is the Doppler velocity error, and  $\mathbf{e}_k$  is the IMU error. The error terms are weighted by the covariance matrix  $\mathbf{W}_k$  in the case of the IMU errors and the normalized intensity of the corresponding radar target

$$w_{d,k} = \frac{i_{d,k}}{\sum_{j \in \mathcal{D}_k} i_{j,k}}, \quad (9.12)$$

in the case of the Doppler velocity measurements where  $w_{d,k}$  is the weight for target  $d$  in scan  $\mathcal{D}_k$  and  $i_{d,k}$  is the intensity of target  $d$ .

In this example, we consider radar measurements consisting of a set of targets  $\mathcal{D}$ . For each  $d \in \mathcal{D}$ , we measure  $[r \ v \ \theta \ \phi]^\top$  representing the range, Doppler (radial) velocity, azimuth, and elevation for target  $d$ . The Doppler velocity measurement  $v$  is equal to the magnitude of the projection of the relative velocity vector between the target and sensor  $\mathbf{v}^s$  onto the ray between sensor origin and the target  $\mathbf{r}^s$ , both in the sensor frame. This is simply the dot product of the target's velocity in the sensor frame and the unit vector directed from the sensor to the target:

$$\mathbf{v}^{s\top} \left( \frac{\mathbf{r}^s}{\|\mathbf{r}^s\|} \right). \quad (9.13)$$

In this approach, it is assumed that the targets in the scene are stationary and only the sensor platform is moving. In this case, each radar target can provide a constraint on our estimate of the sensor rig's velocity in the body-frame. The velocity error for each radar target is then

$$e_{d,k} = v_{d,k} - \mathbf{v}_k^{s\top} \left( \frac{\mathbf{r}_{d,k}^s}{\|\mathbf{r}_{d,k}^s\|} \right), \quad (9.14)$$

where  $v_{d,k}$  is the measured radial velocity of target  $d$  at time  $k$ ,  $\mathbf{v}_k^s$  is the sensor velocity at time  $k$ , and  $\mathbf{r}_{d,k}^s$  is the vector from the sensor to the target  $d$  at time  $k$ . As previously noted, radar measurements are affected by non-Gaussian noise and

radar scans often contain false target data. These challenges may be addressed by using the Cauchy robust norm with the Doppler residual.

### 9.2.2 Direct Odometry

Methods that operate on raw radargrams as the ones depicted in Figure 9.6 and Figure 9.2, as opposed to points filtered from the radargrams, are referred to as *direct* methods.

Direct approaches make use of classical signal processing techniques such as phase correlation and the Fourier-Mellin transform [176, 845]. Given two sequential polar radargrams, their relative rotation can be found by a translational shift in the polar coordinate frame —where a vertical shift corresponds to a change in azimuth angle. Using phase correlation, the relative orientation is selected from the pixel shift that maximizes the agreement of the two polar images. Subsequently, the translation can be refined by a similarly computing the correlation between the images in a Cartesian frame (Figure 9.6). These direct correlation-based methods assume that power returns from a specific location remain stationary over time, enabling meaningful correlation. However, this assumption often fails, especially with dynamic objects or in radar data, where noise artifacts are common.

Correlation is also used in the “Masking by Moving” method of Barnes et al. [53]. Two-dimensional correlation between the current scan and rotated copies of the previous scan is computed on a regular grid of pose candidates. However, Barnes et al. address the problem of nonstationary power returns by training a convolutional neural network (CNN) to avoid including false features that are due to noise. The CNN is trained to predict a mask that keeps only those parts of the radargram that are stationary and thus more useful for correlative scan matching.

In contrast to these direct odometry methods that use all or mostly all of the radargram, the methods in the remainder of this chapter are *indirect* in that they first select specific features or key points and operate on those sparser points to estimate the odometry.

### 9.2.3 Feature-based Odometry

Given that radargrams from 2D spinning FMCW radars are essentially birds-eye-view images, it is natural that several works utilize image-based feature extraction and matching techniques from the computer vision community to find correspondences and estimate odometry; such as SIFT [140, 648], SURF [473], and ORB [545] features. Callmer et al. [140] match large-scale features of islands in an archipelago and Li et al. [648] extract features from a satellite radar. Compared to camera images, the noise level is higher in radar data and highly dependent on the environment. Hence, extracting descriptors that can be used for feature matching is

more difficult [474]. Feature descriptors may also be more place-variant in radar compared to other sensors, making it difficult to do data association if the sensor pose is different. (See also Section 9.3.) FSCD and BASD [905, 980] are examples of key-point extractors and feature descriptors specifically designed for radargrams.

The techniques above involve extracting a set of salient key points (which can reliably be detected in subsequent frames) and computing a feature descriptor describing the surrounding region of the feature point. Given the extracted feature set, the correspondences are computed using feature descriptor matching, often combined with a robust estimator, such as RANSAC. Given the correspondences, the spatial distance between features is minimized, often by finding the least squares solution using Singular Value Decomposition (SVD). In general, a feature-based approach is more stable towards large initial errors compared to the registration based approaches discussed in the next section, since feature descriptors can be associated robustly compared to registration methods that primarily hinge on point proximity for data association.

Going beyond hand-crafted feature descriptors as in the examples above, key-point extraction and feature descriptors can also be generated via deep neural networks, thus allowing features to be automatically generated [52]. One drawback is that training data including radargrams along with ground truth poses from a somewhat similar environment are required.

#### ***9.2.4 Registration-based Odometry***

There are well-developed methods for odometry estimation based on point clouds from LiDAR ranging sensors and similar techniques can be used for point clouds extracted from radargrams or radar datacubes. In this section we describe how such radar point clouds can be computed and used for odometry estimation.

For spinning radars that provide raw signal data as shown in Figure 9.6 we first need to select which points to use by filtering those signal returns that do not correspond to a relevant peak.<sup>2</sup> As discussed in Section 9.1.4, many approaches extract a set of range readings per azimuth; *e.g.*, using CFAR (Figure 9.7, Section 9.1.4), using noise statistics to remove redundant or noisy readings [158], BFAR [25], or simply the  $k$  strongest returns per azimuth. An exception is Kellner et al. [549], using DBSCAN clustering so as to also consider neighboring azimuth angles instead of restricting the search for targets along one azimuth dimension at a time. Models trained with machine learning so as to estimate a point cloud similar to that from a LiDAR from a radargram are also commonly used in recent methods [195, 1207]. A key challenge is to extract an adequate amount of readings; too few readings discards relevant information and too many include noise [158]. For example, CFAR has been found difficult to tune in this respect [127].

<sup>2</sup> These filtering techniques can be seen as similar to the key-point extraction in Section 9.2.3 but without extracting descriptors.

Once a point cloud has been extracted using one of the techniques above, it can either be used as-is or additional information can be estimated by examining the local surrounding region, such as normals, planes and point distributions. The registration approaches used for radar data are often similar to what is done using LiDAR based scan registration approaches (see Section 8.2); however, the noise level and sparsity in radar data makes pair-wise registration much more challenging.

A common strategy in registration-based odometry methods using radar data is to register new scans to multiple previous scans —either aggregated into a submap or as a set of individual point clouds. Registering to multiple scans is a way to compensate for several of the challenges in radar data as discussed above. By including more key frames, the odometry estimate is less sensitive to sparse and noisy radar point clouds. More correspondences adds more constraints which can reduce drift in feature-poor environments. Another goal is temporal redundancy, in the sense that sudden occlusions or spurious correspondences from moving objects impact the odometry estimate less when multiple key frames are used.

Some examples of registration-based radar odometry methods include continuous-time ICP [127], power-shifted NDT [613], and CFEAR [8] (not to be confused with the CFAR method for filtering which is discussed in Section 9.1.4) — all of which make use of submaps or multiple key frames. In CFEAR, point clouds are extracted from radargrams by selecting the  $k$  strongest returns along each azimuth. Each new cloud is registered jointly against the  $s$  most recent key frames using either a point-to-point, point-to-line, or point-to-distribution error metric —akin to NDT scan registration (see Part I). For each point, the normal vector is estimated from the covariance matrix of neighboring points within some radius. Point correspondences are weighted based on the agreement of their normal vectors, the planarity (condition number of the covariance matrix), and the number of points in the neighborhood. Kung et al. [613] use a fixed threshold to extract a point cloud from the radargram and aggregate multiple point clouds into a radar submap using an NDT representation where the contribution of each point is weighted by its returned signal strength. Burnett et al. [127] extract point clouds from radargrams using BFAR [25] and aggregate into a local submap, after which a continuous-time ICP formulation is used to optimize a trajectory estimate where points are associated with a Gaussian process motion prior.

The methods above are all based on 2D radar data. Recent pipelines for registration-based 3+1D radar odometry tend to adopt similar strategies —although point cloud extraction is performed on the sensor so the design choices for selecting which signal peaks to consider as valid targets come down to sensor-specific thresholds rather than explicit feature extraction. Since per-point Doppler speed information is available in 3+1D point clouds, these methods typically consider a least-squares estimate of the ego-velocity from Doppler data (see Section 9.2.1) as an initial estimate and perform point cloud registration to refine it. The registration-based odometry component in 4DRadarSLAM [1268] uses a variant of GICP [982] that is adapted for

*CFEAR*

radar point clouds, where points are weighted by a covariance matrix that assigns a higher uncertainty to points far from the sensor due to the limited azimuth and elevation angle accuracy. 4D iRIOM [1307] employs one-to-many distribution-to-distribution matching in order to alleviate the noise and sparseness of radar point clouds. Instead of matching each point to its closest corresponding distribution in the local submap, each point is matched to a weighted set of closest distributions. The complete SLAM pipelines presented in [1307, 1268] are further described in Section 9.4.2. The EFEAR-4D method [1196] extends CFEAR 2D odometry [8] to 3+1D radar point clouds. After computing a Doppler-based ego-velocity estimate and removing outlier points that do not agree with this least-squares estimate and therefore can be assumed to come from moving obstacles, the remainder of the registration scheme is similar to CFEAR: registering scans to a sequence of preceding key frames and using agreement of normal vector and planarity for associating and weighting individual point matches.

### **9.2.5 Motion Compensation**

As discussed in Section 9.1.3.3, it is important to compensate for motion distortion in odometry estimation. In the case of a low-speed spinning radar with a frequency of 4 Hz, egomotion compensation is reported to reduce ATE (Absolute Trajectory Error) by 29% by using a constant velocity model [8]. Given the time stamps of two subsequent radar scans and the relative pose computed using the methods above, a velocity can be computed and each radar point can be shifted accordingly, since the per-point timing is also available. Offsetting the points of individual radar scans in such a way compensates for the substantial motion that can be encountered during the slow sweep of a scanning radar. The same model is also used to provide an initial estimate to rigid scan registration, and after registering each motion compensated scan to the previous, it is added as a node in the SLAM pose graph.

However, this approach still works in a discrete pose graph setting. Continuous-time trajectory representations can be beneficial for obtaining a smooth and accurate trajectory where the pose estimate used for undistortion can be queried at the time of each sensory reading. In Ng et al. [800] a spline representation is implemented in a pipeline that uses automotive SoC radars. Gaussian processes are used in Burnett et al. [129] to form a factor graph representing the trajectory by combining an IMU sensor with a spinning radar.

## **9.3 Radar Place Recognition**

As with other sensor modalities, place recognition (PR) is an essential module for radar SLAM. A good overview of the problem in general can be found in Chapter 8. As discussed in the chapters on visual and LiDAR PR, securing invariance to

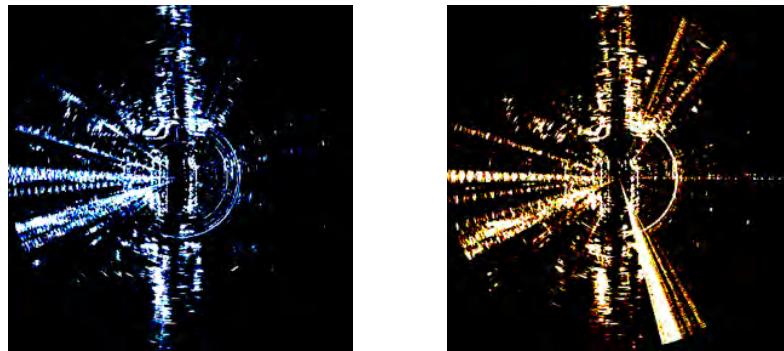


Figure 9.10 Two radar scans obtained from the same location. The noise pattern induces visual aliasing, complicating the process of place recognition.

both translational and rotational change is crucial, such as when traversing a street in a different lane or arriving at a junction from another street.

### ***9.3.1 Unique Challenges in Radar Place Recognition***

While, in principle, several of the methods for PR discussed in the earlier chapters could be applied to radar data after proper data format conversion, some unique characteristics and challenges exist (see also Section 9.1.3).

Several factors contribute to the key challenges in radar place recognition . Firstly, the low resolution of radar data results in less details and thus fewer distinguishable features for recognition. The wide beam of radar data contributes to angular ambiguity, causing distant objects to appear as broad patches that differ significantly when viewed from closer ranges. Additionally, a relatively low signal-to-noise ratio poses challenges for place recognition if adequate filtering is not applied, as demonstrated in Figure 9.6. Receiver saturation can produce a strong radial feature that varies significantly with the observation angle of a particular target, as illustrated in the sample image in Figure 9.10. Consequently, the appearance of a place can change notably from nearby positions due to these challenges.

As there are different types of radars, place recognition needs to be handled differently based on the type of sensor. How a place is perceived and described significantly differs between a long-range 360-degree spinning radar and a SoC radar with limited range and FOV. For example, a spinning radar produces a 2D radargram, which can be treated as an image. Naturally, approaches inspired by visual PR have been applied for the spinning radar. Applying a target detection algorithm such as CFAR to the radargram results in a point cloud, after which methods from 2D LiDAR PR can be adapted. Spinning radars tend to have a very long range which can be greatly beneficial for place recognition [566] yet their slow scanning rate may lead to motion distortion even at low driving speeds. SoC radars, on the

other hand, have a faster update rate since they need not mechanically spin the antenna in order to cover their field of view, and are therefore less susceptible to motion distortion. Most methods that use SoC radar data for place recognition work with a 2D/3D point cloud data format and not the full datacube. As a result, SoC radar PR often builds upon existing LiDAR PR methods; however, the measurements from SoC radars are typically restricted to a small FOV and tend to have higher sparsity and noise.

Currently there is more PR literature using spinning radars than SoC radars. This is in part because spinning radars capture richer information over a wider FOV, allowing them to better capture surrounding structures for robust PR, especially for outdoor applications, and in part due to the availability of large-scale datasets. However, this advantage does not prevent SoC radars from being used for PR. While spinning radars offer a larger FOV, their projection model only provides 2D information, losing elevation details. For indoor PR, where a shorter range and smaller FOV are sufficient, the 3+1D measurements from SoC radars can still be advantageous.

### ***9.3.2 Learning-based Radar Place Recognition***

For spinning radars, treating the 2D radargram as an image, 2D image retrieval has been leveraged for place recognition.<sup>3</sup>

Safteescu et al. [960] is an early approach to PR from 2D FMCW radar which uses a CNN to represent radargrams, specifically addressing their polar nature by using cylindrical convolutions in order to learn a representation that is rotation invariant. Then, a query radargram can be matched against a database of reference images to produce an appearance-only topological PR system. De Martini et al. [246] add pose refinement to produce a topometric mapping and localization system.

PR methods may be trained with augmented instances of the input data in order to be more robust to slight variations. Given that PR data typically is recorded sequentially, scan by scan while driving along a path, “augmented” instances can be retrieved by sampling frames that are sequentially nearby and artificially adding rotation by shifting the polar radargram. Contrasting to such augmented instances, the network is trained not only to recognize similar instances but also to distinguish instances (and their augmentations) that are sequentially far away. In this way, data for training a PR algorithm can be obtained in an unsupervised way, without knowing the true metric location of the radar data [352].

The methods mentioned above have been designed specifically for spinning radar providing 360-degree long-range coverage. SoC radars, which are more attractive for automotive applications given that they lack moving parts and have a smaller

<sup>3</sup> A more comprehensive overview can be found in [121].

size, require slightly different treatment since they typically have lower range and smaller field of view.

Cai et al. [139] use a deep spatiotemporal encoder (after projecting the point cloud to a 2D image plane) to generate feature vectors as place descriptors for topological PR. These vectors are passed through a NetVLAD [35] layer after being re-ranked based on the RCS to filter out non-relevant stationary features. In this case, multiple radar sensors are mounted around the vehicle to overcome the limited field of view.

Herraez et al. [454] demonstrate single-scan radar place recognition with a pipeline that addresses data sparsity and noise by learning to focus on salient points that are important for place recognition. Similar to Cai et al. [139], Herraez et al. [454] also leverage NetVLAD to generate a feature encoding. However, they capture 3D contextual information by using rigid kernel point convolutions, as opposed to projecting the 3D point cloud to a 2D image. Furthermore, a ‘point importance estimator’ outputs the probability of a point being important for place recognition. This estimator is trained with sets of known corresponding point clouds, and query points that have a correspondence within a small radius in the other scan are labelled important. RCS information is incorporated through an separate network that encodes RCS data from the points into a more compact feature representation. Peng et al. [855] filters noise points in a similar way as the Doppler odometry methods in Section 9.2.1. Using RANSAC to estimate the ego-velocity produces a set of inlier points which are likely to be stationary. The remaining outlier points can then be filtered as noise. Rather than using NetVLAD which was designed for visual place recognition, Peng et al. [855] demonstrate a radar-specific feature extraction backbone named MinkLoc4D which takes inspiration from LiDAR place recognition architectures [1317].

### 9.3.3 Descriptor-based Radar Place Recognition

Using hand-crafted descriptors instead of learned embeddings is often effective as well. A common approach for radar PR is to directly adopt LiDAR descriptors, such as Scan Context [567], RING [1209], or M2DP [445]; though proper adjustments are necessary due to the differing sensor data formats. For example, the 3D structural information (*e.g.*, height) is missing in spinning radar data, but can be replaced with RCS. Furthermore, additional care should be taken with radar sensor data to address its inherent challenges, often requiring careful noise filtering, sparsity handling and motion compensation.

Hong et al. [474] implement place recognition by adapting the M2DP descriptor [445] originally designed for 3D point clouds to 2D point clouds extracted from radargrams. Additionally, they investigate the distribution of points on the 2D plane to assess if a point cloud is likely to be distinctive or not. Performing PCA on the 2D points produces two eigenvalues that describe the spread of the points along each

eigenvector. Point clouds where the eigenvalues are substantially different indicate cases where the scan lacks features in one direction (such as data from highway driving) and those scans are not considered for place recognition by Hong et al.

Jang et al. [510] modify the RING descriptor [1209] for radar. The descriptor generated by RING is in a sinogram form providing roto-translation invariance. The correlation between two sinograms should give the correct match for the PR problem; yet, the high level of noise in radar images may prohibit a naive comparison. Additional incorporation of auto-correlation has been shown to enhance the PR performance for radar images.

Adolfsson et al. [9] adapt Scan Context descriptors made from 2D radargrams in several ways. Firstly, they compute the sum of intensities for all points in a Scan Context bin as a way to encode both the intensity and the point density. Further, each descriptor is generated from an aggregated set of noise-filtered and motion-compensated polar images in order to mitigate some of the challenges listed in Section 9.3.1. Keeping only the  $k$  strongest returns along each azimuth direction provides a conservative filter that tends to remove a large part of the noise otherwise present in radar point clouds. Creating the Scan Context descriptor from multiple registered point clouds further addresses the sparse data remaining after this conservative filtering. De-skewing the point cloud via a constant acceleration model is important for generating comparable descriptors when using spinning radars while driving.

PR methods designed for 3+1D SoC radars also commonly implement a variant of the Scan Context descriptor [1268, 1307, 656]. However, as spurious radar points can easily distort the height measurements, using the maximum height per radial bin (as in the original Scan Context) is not always as effective for radar point clouds as for LiDAR. The Intensity Scan Context descriptor [1148] is an alternative that stores the maximum measured intensity value of the points in a Scan Context bin rather than the height. Alternatively, the sum of intensities within a bin can be used, so as to use both the intensity and the point density, thus being able to encode vertical structures in the descriptor without being as susceptible to noisy point positions. Another important factor when used with 3+1D radar is that the modified radar descriptor should cope with a much narrower FOV compared to the 360° LiDAR that Scan Context was designed for. Given that a place will appear quite differently when observed from two different viewpoints, it is difficult to achieve rotation invariance when the sensor only covers a small FOV. One way to address this is to apply loop pre-filtering based on the current odometry estimate, only attempting to match the current descriptor to those of frames within a certain range of yaw angles (*e.g.*, 20°) [1268].

## 9.4 Radar SLAM

Radar SLAM systems generally implement the same overall structures as LiDAR- or camera-based SLAM solutions. In this section we briefly describe notable systems from the past two decades with a focus on aspects that are particular to tailoring SLAM to radar data. We also describe multi-modal systems that combine radar with other exteroceptive sensors in Section 9.4.3.

### 9.4.1 Map Representations

Many radar SLAM systems generate maps that rely on similar representations discussed in Chapter 5. However, the sparse and often spurious nature of radar measurements introduces unique challenges in the mapping process. Some earlier works in radar mapping establish a map representation directly from existing target detection models, often using landmark maps where individual detected targets constitute the map [224, 271, 140, 980]. This target detection can also be used for occupancy grid map as in the series of works by Mullane et al. [778], while exploiting the detection probability [779] in the mapping phase.

A detected target, represented by an individual peak in the signal, can also be treated the same way as a point from a LiDAR scan. For instance, these points can be used to create 2D occupancy grid maps [727, 759] or point cloud maps [474]. The extracted point cloud can also be augmented with additional information, as in [9] which also computes the distribution of surrounding points to estimate orientation and weights, similar to surfels or NDT cells. Direct point cloud representations are also popular in emerging high-resolution SoC radars, which feature a larger vertical field of view (3+1D) and a larger number of TX/RX antennas (*e.g.*, 48 TX + 48 RX antennas for the Sensrad Hugin radar) [1268, 1307].

While less common, the full radar heatmap – comprising intensity samples for each direction and range bin prior to peak detection – can also be used to provide a dense grid map [947]. In this case, the map represents a global heatmap of the reflected power at each point in the environment, rather than evidence of occupancy. The 2D alignment between these heatmaps is then achieved through correlation.

Kramer and Heckman [603], in addition to generating odometry, presented a novel sensor model for voxel based mapping of radar data, capable of creating sparse maps even through visual occlusions. The sensor model leverages the log-odds based estimation of occupied vs free cells used in Octomap [478], but replaces their ray-cast model. The Octomap ray-cast model assumes that the first contact of a sensor is the only relevant point of occupation, but the generalized model accounts for radar’s ability to penetrate certain material types by updating voxel probabilities within the sensors field of view, increasing probabilities in cells with radar returns and decreasing probability with missed scans, without assuming information along a ray. A related grid-map representation specifically designed with radar in mind is due to

Nuss et al. [812] who designed a state estimation filter to address dynamic obstacles in grid maps called a probability-hypothesis-density multi-instance Bernoulli filter. This filter casts grid cells as a finite stochastic set, and fuses radar and lidar data dynamically.

Lastly, the challenges of lower density and higher noise in radar data can be addressed using machine learning techniques [1207, 770], where LiDAR data serves as ground truth to achieve higher resolution and reduced noise in radar outputs. Mopidevi et al. [770] build a global radar map, where patches of the map are upscaled using a predictive network that filters noise from free-space regions and fills in sparse and empty regions, generating a map more similar to what can be obtained with LiDAR data.

Recently, neural fields, originally developed for RGB data [762] and later applied for LiDAR data [1290, 1053] have been applied to 2D radar data as well [98]. The key feature of neural representations is that they implicitly represent the environment such that the neural network can be queried with a point in space and return a quantity such as the distance to the closest surface, the colour and opacity, etc. In the radar fields of Borts et al. [98], a physics-informed radar sensor model as used to create an implicit neural geometry and reflectance model which can then be used to synthesize radar measurements from unseen view points. The received power at the radar sensor depends on the known transmit power and antenna gain but also the RCS which is composed from the size, radar reflectivity and directivity of the object. The neural representation learns to decompose the measured RCS into size (area) on the one hand and the product of reflectivity and directivity on the other hand.

#### **9.4.2 Radar SLAM Pipelines**

In the preceding sections we have covered the main components that make up graph-based radar SLAM frameworks: open-loop odometry estimation, place recognition for loop closure detection, and map representations —in addition to some of the physical and technical principles that are pertinent to radar SLAM.

This section reviews a number of complete SLAM pipelines that use radar as the only exteroceptive sensor and discusses how they implement the components. These pipelines generally follow the same architecture as shown in Figure 9.1, with a front-end that has a filtering process to the raw radar data into a point cloud and a place recognition module to identify loop closures and add the corresponding constraints to the underlying pose graph , and a back-end mapping module that performs frame-by-frame odometry and SLAM-proper module that globally optimizes the map when loops have been detected.

Working with 2D radargrams from a spinning radar, the TBV-SLAM (“trust but verify”) pipeline [9] builds upon the CFEAR 2D radar odometry method discussed in Section 9.2. The pose of the sensor is tracked using every radar scan in

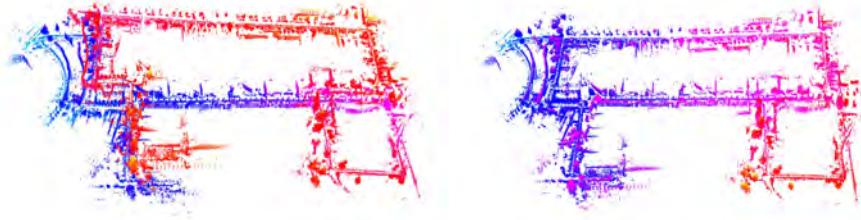


Figure 9.11 Example 3D map produced with radar SLAM using 3+1D radar (Sensrad Hugin) and IMU input. Color denotes height. *Left:* before loop closure. Note the accumulated horizontal and vertical drift that is evident in the left part of 3D map. *Right:* after loop closure.

sequence but in the interest of efficiency, a sparser set of key frames is included in the SLAM pose graph. Once the estimated traveled distance exceeds a certain threshold (*e.g.*, 1.5 m) a new key frame is added to the pose graph and an odometry constraint is created based on the alignment to the latest key frame. As usual, a constraint in the graph requires both the relative pose offset between the two nodes and the associated uncertainty expressed as a covariance matrix. Interestingly, it has been shown [9] that using a predefined diagonal covariance matrix with small values performs better than estimating the covariance based on the Hessian of the registration cost function, which might otherwise be expected to better capture the uncertainty stemming from the shape of the input point clouds such that a pair of point clouds from a tunnel would give a larger uncertainty (along the tunnel’s direction), for example. However, using the Hessian tends to under- or overestimate the uncertainty which may cause the back-end optimization to slightly misalign the key frames. A central part of the TBV-SLAM pipeline is the place recognition module, where several candidate loop closures are retrieved (“trusted”) and later tested after which the verifiably best candidate is selected. The Scan Context descriptor [562] (see 9.3.3) is adapted to account for both point density and signal strength (in lieu of the height data that is unavailable in 2D radar). Additionally, several techniques are implemented for retrieving and verifying loop candidates. For each key frame, several augmented descriptors are created by shifting the point cloud by lateral translation offsets. When searching for loop closures, the query descriptor is matched to all the augmented descriptors currently in the database, in order to account for loop closures where the vehicle is driving in a different lane. Loop candidates found by matching descriptors in this way are then filtered based on the odometry estimate (which hinges on having accurate enough odometry over large distances). After propagating the uncertainty from the odometry constraints between the query and candidate, candidate loop closures where the two scans are estimated to be far apart can be discarded. However, *jointly* considering

the descriptor similarity and odometry uncertainty further improves the robustness of loop retrievals. That is, instead of finding the most similar candidate  $c$  to a query descriptor  $q$  such that  $c = \arg \min_c d_{\text{descriptor}}(q, c)$  and then filtering based on odometry, the candidate is found from  $c = \arg \min_c d_{\text{descriptor}}(q, c) + d_{\text{odometry}}(q, c)$ , where  $d_{\text{odometry}}$  is computed as the likelihood of frame  $q$  being at the same place as  $c$  taking the accumulated odometry uncertainty into account. Pairs of radar scans thus found are then aligned with the CFEAR registration module and finally, an alignment *verification* module which includes overlap measures as well as the CorAl [7] measure trained to detect slight misalignments. All in all, this pipeline demonstrates a number of techniques used to adapt descriptor-based place recognition to radar data (taking into account the multiple signal returns available in a 2D radargram and compensating for the comparatively sparse and slow scanning) and to sift through multiple loop candidates in order to verify the best ones based on geometric alignment as well as the front-end pose estimate.

The RadarSLAM pipeline of Hong et al. [474] is another prominent example of radar SLAM with 2D spinning radar data. In this pipeline, odometry (open loop pose tracking) is achieved by tracking key points detected directly in the radargram. A blob detector generates key points which are then tracked from frame to frame with a Lucas–Kanade tracker [704]. From the traveled distance computed by the tracker, a constant velocity model is used to compensate for the motion during one revolution of the scanner, and transformed key points are stored in the factor graph together with the poses of the key frames. Key frames are, as above, selected based on traveled distance. While pose tracking is done with a sparse set of key points, for loop closure detection denser point clouds are extracted from the radargrams. This point cloud extraction is done similarly as in TBV-SLAM [9]; however, instead of selecting the  $k$  strongest points per azimuth, RadarSLAM selects all points with an intensity higher than one standard deviation over the mean value. M2DP descriptors [445] are then created from the point clouds of the key frames, and loop closures are detected by matching frames with similar descriptors. As a safeguard against matching nondescriptive point clouds, RadarSLAM avoids selecting loop closures from key frames that are too elongated; *i.e.*, where the two eigenvalues computed from PCA are markedly different, since such point clouds are expected to be from non-unique places like a highway section. No other loop verification is performed.

One example of a SLAM pipeline based on SoC 2D radar is due to Schuster et al. [980]. Differently to the methods above, they maintain a graph that consists not only of the sensor poses but also individual radar feature nodes, whereas TBV-SLAM and RadarSLAM only optimize a graph of poses (although radar points are associated to the pose nodes in order to facilitate place recognition and rendering of the map). As 2D SoC radars generally provide far fewer detections than 360-degree spinning radars or 3+1D SoC radars, maintaining all detections in the optimizable graph is more feasible here. Edges are added in the graph to represent observations between all concurrently observed features. Their landmarks are extracted using a

binary annular statistics descriptor (BASD [905]). As BASD is a compact binary descriptor, it is feasible to directly compare the descriptors of all feature points in a local region so as to associate recent features with those already in the map. Those features that pass a RANSAC outlier rejection stage are added as vertices to the graph, along with a pose node with odometry information from wheel encoders. Assuming moderate drift from open-loop tracking, no explicit place recognition step is included, but point features can be matched with the BASD descriptors after loop closure, after which the SLAM graph is optimized in the back-end.

Two recent approaches to 6DOF radar SLAM with 3+1D SoC radar are 4DRadarSLAM [1268] and 4D iRIOM [1307], both using 3D radar point clouds as input, where the point detection (filtering of the datacube) is handled onboard the sensor itself, so CFAR or other peak detection is not explicitly included in the SLAM pipeline. Still, the input point cloud may contain a lot of noise points. The Doppler radial velocity information included in the 3+1D point clouds is exploited by both methods to filter points from moving objects. The vehicle's ego-velocity can be estimated from linear least squares of the measured Doppler point velocities, and outlier points for which the velocity model does not agree are removed. iRIOM further denoises the point clouds by keeping as inlier points only those that have sufficiently many neighbor points (within a fixed radius) and where those points are compactly distributed (considering the covariance matrix of their spatial distribution). The Doppler ego-velocity estimation is used as a prior to a scan-to-submap registration step. 4DRadarSLAM uses a variant of GICP [982] termed APDGICP where points are weighted by a covariance matrix that assigns a higher uncertainty to points far from the sensor due to the limited azimuth and elevation angle accuracy. The submaps (key frames) are inserted into a graph. Loop closures are detected by Scan Context matching, and as opposed to the 2D methods above, the original Scan Context descriptor that includes point elevation can be used. While 4D iRIOM uses the original Scan Context, 4DRadarSLAM uses Intensity Scan Context [1148] in order to avoid uninformative descriptors due to noisy elevation measurements. 4DRadarSLAM additionally includes a validation step to reject candidate matches returned by Scan Context if the accumulated odometry distance between the two frames is above a certain threshold.

#### 9.4.3 Multi-modality in Radar SLAM

So far we have mostly been concerned with methods for radar SLAM that use only radar data and in some cases proprioceptive sensing like IMU or wheel odometry. In this section we discuss systems that combine mmWave radar with other exteroceptive sensors (*e.g.*, LiDAR, camera) or external data (satellite imagery or prior maps).

Some works in radar mapping have employed radar–LiDAR fusion so as to generate the best possible set of points given the current visibility conditions (trusting

LiDAR more in clear conditions and radar more in low visibility). Fritzsche et al. [342, 343] fuse the sensors based on estimated ranges to determine which sensor to trust in a given case. Radar and LiDAR have also been combined to improve place recognition, overcoming different sensor modalities by registering radar to LiDAR maps [1244, 1245].

Doer and Trommer [273] extend ROVIO [90], which is a filter-based visual-inertial odometry approach, to integrate radar egovelociy estimates using the Doppler odometry approach described in Section 9.2.1 [272]. In a similar way, also thermal camera data can be fused with radar to achieve a multi-modal radar-thermal estimation pipeline [273]. Zhang et al. [1269] combine data from thermal camera and a 3+1D radar point cloud in order to get robust frame-to-frame odometry in low-visibility settings. A transformer-based feature matcher detects corresponding points in sequential thermal frames and the radar point cloud is used to improve the depth estimate.

In terms of using multi-modal data for radar-based navigation, it is also worth mentioning methods that make use of overhead images and road maps, although most works in the literature exploit this kind of data specifically for localization, rather than for full SLAM. Hong et al. [475] demonstrate how 2D scanning radar data can be used to localize in prior public maps such as OpenStreetMap. Their system runs odometry using RadarSLAM [474] and represents the estimated pose of the sensor as a Gaussian mixture model. Line segments corresponding to building walls are extracted from OpenStreetMap which is used as a prior. Oriented points that are extracted from the radargram are then matched to the features of the prior map. However, given that the prior map information is uncertain and may be incomplete or outdated, this point-to-feature data association is challenging. Poses are sampled from the Gaussian mixture model of the current pose estimate and for each pose the oriented points of the current radar scan are matched to the line features of the prior, which localizes the scan to the prior. Another method that uses prior map data, in this case satellite imagery, for localising 2D radargrams is RSL-Net [1066], which consist of a set of deep neural networks. The first generates a synthetic image from an input overhead photo, showing what a radargram from that place might look like. Another networks estimates the relative rotation between a real radargram and an overhead photo (via the synthetic radargram generated in the previous network). Finally, another network estimates the relative translation offset between the radargram and the overhead image.

Some systems make use of so-called ultra-wideband (UWB) radio sensing. While UWB also uses electromagnetic waves within the radio spectrum and is sometimes referred to as UWB radar, the ranging capabilites of UWB differ dramatically from mmWave radar. When used in a radar SLAM framework, UWB radar is mostly used to detect similarities between sensor readings from different places. The frequency reponse after sending out a wide-band and wide-beam signal can provide a signature of the current location. The metric information is instead derived from

wheel encoders or IMU data. Schouten and Steckel [976] and Takeuchi et al. [1062] use a database of UWB wave signatures to detect revisited places. For each place (node), a signature from the radar echo is stored along with the estimated pose. A graph is then created and optimized with odometry and loop constraints. Both approaches rely on odometry to obtain distances between nodes (*i.e.*, edges in a graph) for metric SLAM and the signatures are created using a pulse-echo UWB sensor. Premachandra et al. [883] conversely use UWB radar to detect point features to be used in a landmark-based SLAM framework. They make use of multiple radar modules on each side of the robot and use trilateration of matched peaks in the signals from the sensors on either side to detect landmarks. In addition to the above, several UWB-based localization approaches use anchor-tag sensor configurations, where anchors are fixed to known locations and a battery-powered UWB tag is mounted on the robot or the asset to be localized. However, as this approach requires preinstalled infrastructure it is not directly related to SLAM.

## 9.5 Radar Datasets

In this section, we briefly summarize notable datasets from the radar SLAM literature. The datasets are also listed in Table 9.1.

**Spinning Radar.** The datasets with spinning radar in Table 9.1 all use Navtech 2D radar sensors. Two of the first large-scale radar datasets for odometry and SLAM are the Oxford Radar Robotcar dataset [54] and MulRan [566]. These datasets have both been quite well used in the literature. The Oxford dataset covers a set of traversals of an urban driving route, totaling 280 km in various weather conditions. The MulRan dataset covers a more diverse set of environments, both dense urban and more rural driving, and longer time spans between sessions, but less driving in total. MulRan focuses on facilitating PR research but has also been well used to benchmark odometry and SLAM methods. The Boreas dataset [128] includes data from driving a route repeatedly over the course of one year (385 km in total), notably including adverse weather conditions such as snow and rain. In addition to SLAM-related benchmarks, this dataset also includes benchmarks for object detection (cars, pedestrians, cyclists). The Oxford Offroad Radar Dataset [353] is focusing on non-urban driving, in contrast to the other datasets in Table 9.1. This dataset covers about 154 km driving on unpaved roads and mountain trails in unpopulated areas.

**SoC Radar.** Several SoC radar datasets are also available, both with 2+1D and 3+1D data. Some datasets geared towards autonomous driving focus primarily on object detection but have also been used for developing and testing SLAM approaches. NuScenes [137] combines data from five Continental ARS408-21 radars mounted on the car used for data collection with one LiDAR and six cameras. The dataset focuses on urban driving, in four cities, and notably includes annotated labels for object detection of 23 object classes. RadarScenes [979] is a dataset with

	Dataset	Lidar	Cameras	Ground truth	Environment	Inclement Weather
<b>Spinning radar</b>	Oxford Radar RobotCar [54]	Yes	Stereo/Mono	GPS/IMU + VO	Dense Urban	
	Boreas [128]	Yes	Mono	GPS/IMU + RTK	Sparse Urban	
	MulRan [566]	Yes	No	SLAM	Mixed Urban	-
	RADIATE [998]	Yes	Stereo	GPS/IMU	Mixed Urban	
	OORD [353]	Yes	Mono	GPS	Urban and Offroad	
<b>SoC array radar</b>	nuScenes [137]	Yes	Stereo	GPS/IMU	Mixed Urban and Natural	
	RadarScenes [979]	No	Mono	None	Mixed Urban Roadways	
	ColoRadar [605]	Yes	No	SLAM	Varying	-
	NTU4DRadLM [1270]	Yes	Mono	SLAM	Mixed Urban	-
	MSC-RAD4R [212]	Yes	Stereo	GPS + RTK	Mixed Urban	
	Snail [482]	Yes	Stereo	TLS	Roadways and Tunnels	
	K-Radar [833]	Yes	Stereo	GPS/IMU + RTK	Roadways	
<b>Both</b>	TruckScene [324]	Yes	Stereo	GPS/IMU + RTK	Roadways	
	HeRCULES [568]	Yes	Stereo	GPS/IMU + RTK	Mixed Urban and Natural	

(: Rain : Snow : Night : Fog : Smoke)

Table 9.1 *Overview of public radar-related datasets. In the ‘ground truth’ column, VO denotes visual odometry, TLS denotes survey-grade terrestrial laser scans, RTK indicates GPS with real-time kinematic corrections.*

four 77 GHz automotive 2+1D radars (unnamed) and one camera. It focuses on semantic perception and contains labels for 11 object types, but lacks accurate ground truth as well as IMU and LiDAR data. ColoRadar [605] is a radar SLAM dataset with data from a 3+1D Texas Instruments MMWCAS-RF-EVM board as well as 2+1D Texas Instruments module, in addition to IMU and 3D LiDAR data. Notably, this dataset includes raw analog-to-digital converter (ADC) values from the radar sensors in addition to 3D ‘heat maps’ (data cubes) and individual point targets. It covers both indoor and outdoor data, as well as data from an underground mine, and includes 6-Degree of Freedom (DoF) ground-truth tracking for pose estimation. NTU4DRadLM [1270] and MSC-RAD4R [212] both include high-resolution 3+1D radar data from an Oculii Eagle sensor. NTU4DRadLM covers structured (university campus) and unstructured (park) environments and MSC-RAD4R covers urban and rural on-road driving. The Snail-Radar dataset [482] features two high-resolution 3+1D radars: both Oculii Eagle and Continental ARS548, and includes data from handheld collection and on-road driving in urban environments.

## 9.6 Further Readings & Recent Trends

Radar SLAM pipelines that work in 3D are still rather few but as high-resolution SoC sensors develop we can expect to see more work on fully 3D odometry and place recognition with radar. This will be particularly important on drones, for example, where radar is less utilized today. Creative designs will be required to enable 3D wide field of view (FOV) radar units. In the meantime, we are likely to see significant advancements in handling constellations of several small FOV radars in SLAM. This naturally comes along with calibration concerns and other issues.

There will also likely be a shift towards making better use of the raw radargrams and datacubes, which contain spectral data. Traditional radar SLAM systems often

resort to generating point clouds to interpret the environment. However, future systems are expected to take fuller advantage of the rich spectral information available in radar data, providing more detailed and nuanced maps and improving both object detection and classification. One challenge to this is convincing manufacturers to open up access to the raw output of their products for research.

Radar semantic segmentation may also play a more prominent role in place recognition. By leveraging segmentation techniques, SLAM systems can more effectively differentiate between various types of objects in the environment, allowing for more intelligent navigation and decision-making. This will also help in reducing ambiguities in radar returns, leading to more reliable mapping.

Finally, there will be a greater focus on multi-modal data approaches, which integrate radar data with other sources of information including other sensors and also geographic priors (*e.g.*, OpenStreetMap). For example, by combining radar observations with these priors, radar SLAM systems may be able to localize more accurately in large-scale outdoor environments, further enhancing the robustness and reliability of autonomous systems.

# 10

## Event-based SLAM

Guillermo Gallego, Javier Hidalgo-Carrió, and Davide Scaramuzza

An inquisitive reader would notice that SLAM is paramount in real-world applications that involve interpretation of spatial relationships and interaction with the environment. SLAM’s primary sensors are critical for the system’s success and adaptability. Visual SLAM is one of the most pervasive categories of SLAM methods because cameras are broadly available (affordable) and produce an intuitive and informative signal that allows the robot to sense the world in a wide range of scenarios (*e.g.*, yielding lightweight systems that do not require additional infrastructure like GNSS). Despite the progress so far, state-of-the-art artificial vision systems are not as effective (robust and efficient) in real-world tasks as their biological counterparts. Standard cameras sense the world at a fixed frame rate that is independent of the scene dynamics. Thus, they become blind in the time between frames, introduce latency, potentially lose tracking, and produce large amounts of redundant data if nothing moves in the scene. This chapter pursues the visionary challenge of understanding and building visual SLAM systems that are fast (not limited by a frame rate), low-power, and robust to broad illumination conditions, by leveraging the bioinspired technology of silicon retinas or “event cameras”, which overcome several of the limitations of standard cameras (see Fig. 10.1).

We start by describing the working principles of event cameras (Section 10.1), as well as the corresponding challenges and applications (Section 10.2). Then we focus on methodologies to process event camera data (Section 10.3), and the corresponding front-end (Section 10.4) and back-end processing (Section 10.5). Finally, we discuss state-of-the-art systems (Section 10.6), datasets, simulators, and benchmarks (Section 10.7), and we conclude the chapter with a discussion about new trends and further readings (Section 10.8).

### 10.1 Sensor Description

#### 10.1.1 Working principle

In contrast to traditional cameras, which acquire full images at a rate given by an external clock (*e.g.*, 30 Hz), the pixels of event cameras like the Dynamic Vision

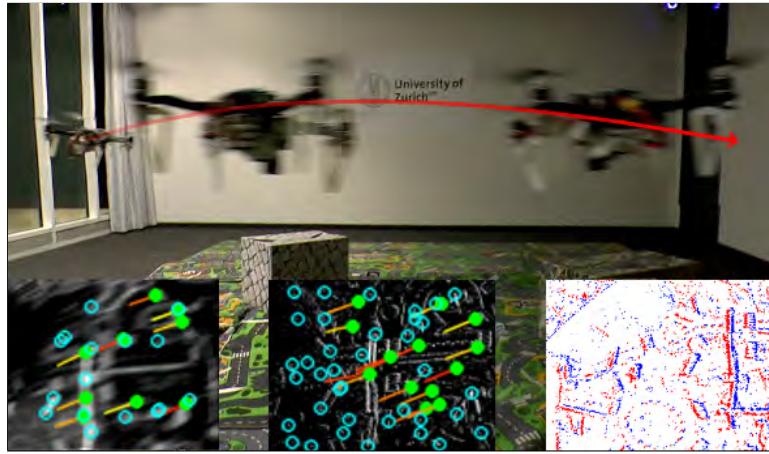


Figure 10.1 Drone with a downlooking DAVIS camera [111] ( $240 \times 180$  px) performing an autonomous flight using a visual-inertial odometry (VIO) algorithm [941] for state estimation. The high speed and high dynamic range characteristics of the event camera data are leveraged to operate in difficult illumination conditions. The insets show features (*i.e.*, keypoints) detected and tracked in grayscale frames (left, motion-blurred) and in motion-compensated images of warped events (middle, sharp). The event data (in red/blue according to polarity) clearly respond to the scene contours. The same VIO algorithm [941] is also demonstrated on high-speed scenarios, such as an event camera spinning tied to a rope. Image from [361] (©2020 IEEE).

Sensor (DVS) [659, 361] operate independently from each other, responding to brightness changes in the scene asynchronously, as they occur (Figure 10.2b). These pixelwise changes are due to scene illumination (*e.g.*, flickering lights) and/or to the relative motion of the camera and the scene (including moving objects). Hence, the output of an event camera is a sequence of digital “events” (or “spikes”), where each event represents a change of brightness (logarithmic intensity). This encoding is inspired by the spiking nature of biological visual pathways (Figure 10.2a).

Specifically, each pixel memorizes the logarithmic intensity  $L$  each time it sends an event, and continuously monitors for a change  $\Delta L$  of sufficient magnitude from this memorized value (Figure 10.2). When the change reaches a threshold  $C$ ,

$$\Delta L \doteq L(\mathbf{x}_k, t_k) - L(\mathbf{x}_k, t_k - \Delta t_k) = p_k C, \quad (10.1)$$

the camera sends an event,  $e_k \doteq (\mathbf{x}_k, t_k, p_k)$ , which is transmitted from the chip with the  $x, y$  pixel location  $\mathbf{x}_k$ , the time  $t_k$ , and the 1-bit polarity  $p_k \in \{+1, -1\}$  of the change (*i.e.*, brightness increase or decrease).  $\Delta t_k$  is the time elapsed since the previous event at the same pixel.

Event cameras are data-driven sensors: their output depends on the amount of motion or illumination change in the scene. The faster the motion, the more events

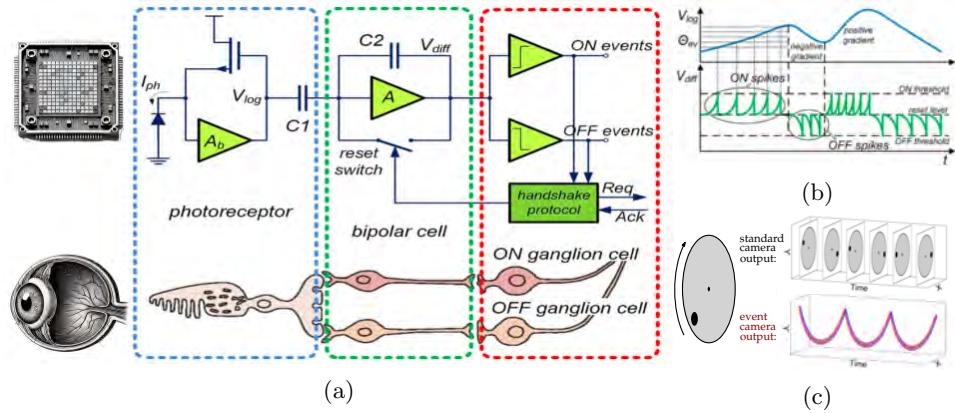


Figure 10.2 Working principle of an event camera (*e.g.*, DVS): (a) Three-layer model of a human retina and corresponding DVS pixel circuit; (b) Schematic of the operation of a DVS pixel, converting light into events (spikes), with the colors of the signals matching those of the layers in (a); (c) Comparison of the response of a standard camera and an event camera to a visual stimulus consisting of a black dot on a rotating disk. An event camera transmits the brightness changes continuously, forming a spiral of events in space-time. Red color: positive events (ON spikes), blue color: negative events (OFF spikes). Images adapted from [881] (©2014 IEEE).

per second are produced because each pixel adapts its sampling rate to the rate of change of the intensity signal that it monitors.

#### *visual pathway*

**Bio-inspiration: The Transient Pathway.** Event cameras are inspired by the operation of biological visual pathways, which are the information processing routes in animals and humans. Following the two-stream hypothesis, the dorsal stream (also called “transient” or “where” pathway) is dedicated to processing dynamic visual information (*e.g.*, motion in the scene), whereas the ventral stream (called “sustained” or “what” pathway) is dedicated to object and visual identification and recognition. The DVS [659] corresponds to the part of the transient pathway from the photoreceptors up to the ganglion cells, adopting a simplified 3-layer pixel design that balances biological fidelity and circuitry stability (Figure 10.2). The three layers realize the functions of light conversion, delta-modulation, and comparison, respectively. Cameras like the Asynchronous time-based image sensor (ATIS) [880] or the Dynamic and Active-Pixel Vision Sensor (DAVIS) [111] model both visual pathways, and therefore output two types of signals: DVS events and grayscale information (*e.g.*, images). More details of the main event camera types are provided in [881, 361].

### 10.1.2 Advantages of Event Cameras

The sensing principle of event cameras is radically different from that of standard (exposure-based) cameras that have dominated computer and robot vision for the last seven decades, and it offers numerous advantages:

*High Temporal Resolution:* events are detected and timestamped with microsecond resolution, which enables capturing very fast motions without suffering from motion blur typical of frame-based cameras. Events are produced almost continuously in time, thus avoiding blind times that can cause large inter-image displacements and ruin data association in standard cameras.

*Low Latency:* each pixel works independently, without waiting for a global exposure time, thus events are transmitted as soon as a brightness change is detected, with submillisecond latency.

*Low Power and Bandwidth:* events represent non-redundant temporal data, hence power is purposely spent. Bandwidth is also reduced (compared to a traditional camera operating at the same rate). At the die level, cameras consume less than 10 mW, allowing embedded systems to consume 100 mW or less [31].

*High Dynamic Range (HDR):* the range of light values that event cameras can sense is very high (typically >120 dB vs. 60 dB of standard cameras), enabling them to sense very dark (moonlight) and very bright (daylight) regions, simultaneously. Hence, they do not suffer from under/over-exposure typical of frame-based cameras. This property is due two facts: each pixel works independently and converts light to voltage in logarithmic scale.

### 10.1.3 Current Devices and Trends

*Which event camera should I buy or use to solve my SLAM problem?* We often get asked this question by people entering this emerging field. The characteristics of event cameras are often compared via tables [361, Table 1], [162, Tables 1–2]. Although multiple event camera designs exist, most of them are laboratory prototypes. Only a few make it into commercialized devices that enable the exploration of novel solutions to classical as well as new problems, such as event-based SLAM. Among the devices commercialized by the main manufacturers (SONY, Samsung, iniVation / SynSense, Prophesee, Omnipixel), some trends are worth mentioning:

*Pixel size:* following the megapixel race of traditional cameras and pressure from industry requirements, the pixel pitch (*i.e.*, size) has considerably decreased, from 40 µm (DVS128 [659]) to less than 5 µm [326]. DVS pixels carry out more operations (modulation, comparison, etc.) than their traditional counterparts; hence, they require more transistors, which are more difficult to pack in the same sensor area. To maximize the area of the pixels exposed to light (that is, the fill factor) and reduce the gap between the photoreceptive parts of the pixels, stacked technology and backside illumination have been adopted [326].

*Grayscale output:* early devices such as the DAVIS or ATIS concurrently output grayscale data (*e.g.*, images [111]), which is especially useful in applications with stationary cameras (albeit this is not the usual scenario in SLAM). Newer models such as HD event cameras [326] discontinued the grayscale output in favor of more area for the event output, driven by the megapixel race.

*Color* is not essential in many motion-related tasks, and therefore only a couple of event camera models offer color filters to detect changes in respective color channels (red, green and blue – RGB) [767].

*Inertial data:* some cameras also provide data from an inertial measurement unit (IMU) integrated in the same device. IMUs are valuable complementary proprioceptive sensors to cameras, enabling visual-inertial odometry (VIO) and SLAM, and yielding higher robustness and accuracy than single-sensor systems.

It is unrealistic to think that high-spatial-resolution event cameras are per se better than low-resolution ones. While capturing fine spatial details is important, noise and bandwidth also play an important role in the target application requirements. In SLAM and related tasks, where event cameras may move fast and/or over high-textured scenes, HD (1 Megapixel) event cameras can produce hundreds of millions of events per second. This poses problems, such as saturation of the output transmission bus of the camera, and high processing demands; currently there is no algorithm-and-hardware combination that can process such event rate in real time (without resorting to array-like conversion and/or sub/downsampling). New hybrid sensors, such as [1235], with lower spatial resolution for events than for intensity output, or foveated sensors [317], mimicking biological vision to decrease bandwidth), are being developed; they may provide alternative solutions to the above issue. In SLAM, a lower pixel resolution (*e.g.*, QVGA) is preferred for algorithm prototyping and for real-time operation on computationally-constrained robots. Often the choice of field of view (optics) is as important as the pixel count.

*hybrid vision sensor*

*sparsity*

## 10.2 Challenges and Applications

Event cameras represent a revolutionary technology in visual data acquisition. Hence, they pose the challenge of designing novel methods (algorithms and hardware) to process the acquired data and extract valuable information from it, unlocking the advantages of the sensor. In particular, the main challenges are:

*Dealing with the space-time output:* The output of event cameras is fundamentally different from that of standard cameras: events are asynchronous and spatially sparse, whereas images are synchronous and dense. Hence, visual SLAM algorithms designed for image sequences are not directly applicable to event data.

*Dealing with motion-dependent data:* Unlike images, each event contains binary (increase/decrease) brightness change information that depends not only on the scene texture, but also on the relative motion between the scene and the camera.

*Dealing with noise and dynamic effects:* Event cameras are noisy because of the

inherent photon shot noise, transistor circuit noise, their dependency on the amount of incident light, non-idealities and low-power (sub-threshold) operation.

These challenges call for new approaches that rethink the space-time, photometric and stochastic nature of event data. In the context of SLAM, this poses questions such as: What is the best way to extract information from the events for pose or depth estimation? What map and camera trajectory representations shall be used that take into account the quasi-continuous temporal granularity and sparse nature of event data? How to establish correspondences (data association) under motion-dependent data? How to model the problem (and its solution) without introducing the typical bottlenecks of frame-based technology?

The above questions have been driving the research on event-based SLAM (Fig. 10.3). This topic has evolved both on its own and in conjunction with other tasks, *i.e.*, research on event-based SLAM has fostered research on other event-based tasks. For example, the synergy between SLAM and image reconstruction (the task of recovering absolute intensity from events) has been leveraged as early as the first works [229, 569] (rotational-motion SLAM) and [570] (6-DoF SLAM). Event-based SLAM and optical flow estimation have been treated together in [229, 1240, 1007, 584].

### 10.3 Overview and Taxonomy of Event-based SLAM Methods

Event-based SLAM methods can be broadly categorized in two classes, depending on how many events are processed simultaneously: (*i*) methods that operate on an *event-by-event basis*, where the state of the system (*e.g.*, scene map and camera trajectory) can change upon the arrival of a single event, thus achieving minimum latency, and (*ii*) methods that operate on *groups / batches / slices / packets of events*, which introduce some latency. A key design choice in the latter category is how to select the size of the packet, for which many solutions have been proposed (*e.g.*, fixed number of events, fixed temporal duration, and hybrid criteria).

Orthogonally, depending on how events are processed, model-based approaches and data-driven (*i.e.*, machine learning) approaches can be distinguished. Mimicking the categorization in frame-based SLAM, event-based SLAM methods can be classified into *indirect* methods (feature-based, using event corners, lines, normal flow, etc.) and *direct* (using all events). This categorization is related to the type of objective or loss function used: geometric- vs. photometric-based (*e.g.*, a function of the event polarity or the event rate/activity), and also to the overall philosophy: indirect methods typically have two steps (a feature extraction step, which “converts” events into geometric primitives, followed by a geometric SLAM pipeline), whereas direct methods typically comprise a single step that maps event data into motion and scene parameters. In the latter, the event generation model (10.1) (or its linearized version [361]) is a cornerstone for designing estimation methods. Handling data association between events is a central problem in event-based vision, and SLAM in particular. Due to the high temporal resolution of event cameras,

*event-by-event*

*event batches*

*direct vs. indirect*

*event generation model*

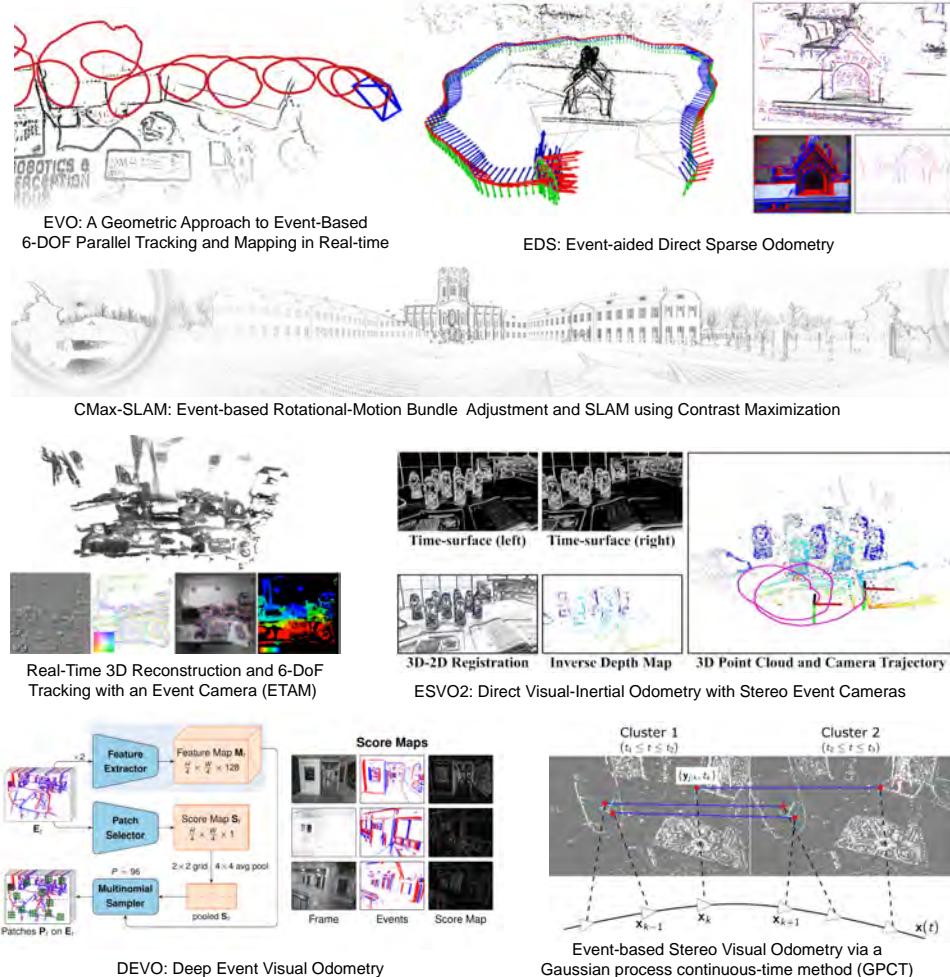


Figure 10.3 Event-based SLAM is actively being investigated, with systems that explore a large variety of approaches, including classical methods and more recent deep learning solutions. Since events are triggered by moving edges on the image plane, it is natural to recover scene maps in the form of edges (e.g., sparse or semi-dense 3D maps). Images adapted from EVO [911] (©2017 IEEE), EDS [461] (©2022 IEEE), CMax-SLAM [416] (©2024 IEEE), Kim et al. [570] (©2016 Springer), ESVO2 [809] (©2025 IEEE), DEVO [584] (©2024 IEEE), and Wang et al. [1151] (©2023 IEEE).

data association is typically handled by temporal and spatial vicinity; both hard-association and soft-association strategies have been explored.

Each of the above categories has advantages and disadvantages. The problem of solving SLAM with event cameras is challenging, and has been historically tackled by gradually increasing complexity along several axes: the number of unknowns

(degrees of freedom – DoFs), the type of motion (from rotational or 2D scenarios to 6-DoF motion), the scene complexity (texture) and its motion (static vs. dynamic – independent moving objects – IMOs). Event-based SLAM is not an isolated problem: as mentioned in Sec. 10.2, it has connections with other problems (optical flow, tracking, segmentation, etc.), in stronger or weaker form depending on the assumptions or scenario considered. In addition to the above-identified trends, it is noticeable that early research has focuses on model-based methods, whereas more recent papers explore the possibilities that deep-learning–based approaches offer.

## 10.4 Front-end of an Event-based SLAM System

Event-based SLAM systems often consist of several modules, which tackle smaller subproblems, such as feature extraction, data association, bootstrapping, pose estimation, depth estimation, etc. A primary division consists of the front-end and the back-end. From an input-output point of view, the front-end receives the raw sensor data (plus possibly auxiliary information, such as camera calibration) and outputs a set of event camera poses and scene map(s) (see Fig. 10.4). The back-end refines these variables (*i.e.*, the SLAM problem unknowns) to improve the fit between them and the sensor data. The back-end operates after the front-end, at a slower pace (depending on the number of variables involved) and can feed back its output to the front-end to help reduce drift and correct errors.

Therefore, the front-end converts the information from the sensor (*e.g.*, photons) into geometric primitives (*e.g.*, camera poses) and also photometric information (*e.g.*, map appearance). This often comprises a step of “feature” or “information” extraction. Hence, the first challenge is to understand the information contained in the event stream and be able to extract it using methods that preserve the characteristics of the data (low latency, sparsity, HDR, etc.). Assuming constant illumination, events are caused by moving contours (edges). Therefore, we may consider a moving event camera as an asynchronous edge detector, which means that the SLAM problem is formulated in terms of scene contours (Fig. 10.3). This is a priori sensible because contours are the most informative regions of the image plane, allowing us to estimate retinal motion, from which 3D information is inferred. Each event consists only of a 4-tuple and is subject to noise, hence it carries little information; thus many events (*e.g.*, thousands, millions) are needed to produce reliable estimates of quantities such as camera poses and scene maps. Extraction of information from the event stream depends on the task and on many design choices, such as the type of representation of the SLAM variables (scene map, camera trajectory), the hardware used to process the data, the output rate, etc.

*geometric primitives*

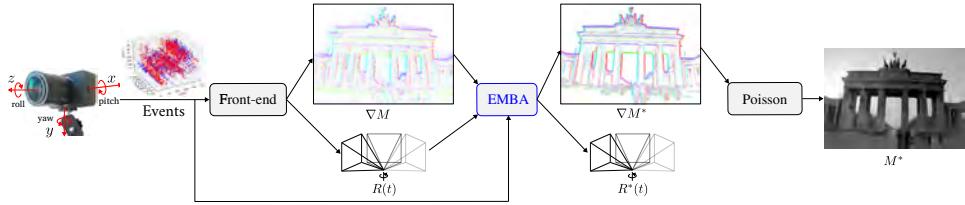


Figure 10.4 Event-based SLAM pipeline with a front-end (that computes and a map and camera poses) and a back-end (that refines the map and poses). Since events respond to moving edges, the recovered map is often an edge (gradient) map. The example shows a direct, rotational SLAM pipeline (poses consists of rotations, and the map reduces to a panoramic map) [417]. An absolute intensity map may be recovered by Poisson integration.

#### 10.4.1 Pre-processing and Event Representations

In the SLAM problem, the event camera continuously outputs data as it moves through the scene. Events are triggered “everywhere” on the image plane, as from the camera’s point of view it appears that all scene edges are moving. Since events are sparse and have microsecond resolution, each of them corresponds to a different camera pose. This is radically different from traditional (frame-based) cameras, where all pixel measurements of an image have the same timestamp and therefore share a common camera pose (this is the paradigm on which traditional multi-view geometry [437] has been built). Many SLAM methods convert event data into alternative representations (event images, time maps or “time surfaces”, voxel grids, etc.) [361] for different reasons, such as compatibility with conventional computer vision methods, easier interpretation, etc. This conversion step often implies a quantization of the information (*e.g.*, grouping events with similar timestamps) and/or a loss of the sparsity (*e.g.*, zero-filling arrays at locations where no events happen).

Therefore, the study of *event representations* [361, 370] has gained attention. It is typically the first stage of the front-end and it highly influences later processing stages: events are converted into a more familiar representation (*e.g.*, images) that are easier to work with (to feed to mature SLAM methods designed for traditional images, or to design learning-based methods based on images). This conversion is in part due to the fact that the research community is still exploring the best way to extract information from the event stream and tries to reutilize mature image-based methods. The front-end may use different event representations; for example, EVO [911] uses raw events for its mapping module (EMVS [913]) and event (edge-like) images for its camera tracking module. Ideally, one would design SLAM methods that use event representations that preserve the high speed and sparse properties of event cameras and do not suffer from the issues of traditional cameras (quantized time, latency, non-sparsity). In practice, this is an emerging research topic that requires rethinking visual processing asynchronously, and there is still ample room for improvement and investigation of fundamental results.

### 10.4.2 Indirect Methods

The design choices of the front-end largely influence the rest of the system. A major design choice is the type of processing method: indirect or direct. Indirect methods have broadly two steps; they first extract and track point-based, line-based or other type of feature from the events, and then leverage results from classical SLAM to estimate the camera motion and the structure of the 3D scene based on such geometric primitives. Features compress the event data into few informative primitives, which enables more effective and efficient use of the computational resources. A central problem consists in establishing and maintaining correspondences among the event features (and the map landmarks), which is known as data association. This is challenging, as each event carries little information and is motion-dependent to unambiguously determine association. Due to the high temporal resolution of event cameras, association can be established by spatio-temporal vicinity in pixel space. Hence, it is natural to track features rather than to match them.

Camera pose estimation or camera tracking is often formulated as the solution of a feature registration / alignment problem by minimization of a geometric objective (*e.g.*, the reprojection error, measured using Euclidean distance in pixel space) given a map of the scene. The 3D structure of the scene is typically computed by means of triangulation (*i.e.*, back-projection) of corresponding feature locations (*e.g.*, to obtain 3D points and lines) using given camera poses. A large toolbox of mature geometric methods (multi-view geometry [437]) can be exploited.

Like conventional visual SLAM, indirect event-based methods rely heavily on robust feature extraction and tracking. However, these components are not yet as mature as their frame-based counterparts because they have to deal with unique challenges (large noise, sparsity, asynchrony, motion dependence, etc.). This limits the accuracy and therefore applicability of these systems. To address these issues, some systems resort to sensor fusion (with grayscale images and/or IMU data).

### 10.4.3 Direct Methods

Direct methods use all data available (not just the event data that conform to the definition of a feature) to estimate camera motion and 3D scene structure. They directly align event data with maps, images or other events without explicit feature extraction. If the event rate is high compared to the processing capacity of the system, data reduction mechanisms (*e.g.*, denoising or subsampling) are adopted to reduce the number of events to process [416, 571].

As direct methods have only one step, the motion (camera tracking) or scene parameters (mapping) are obtained by optimization of some objective function (*e.g.*, photometric error or spatial event rate error). The photometric-based objective induces a geometric registration objective. The problem unknowns are obtained by the alignment of edge-like brightness patterns conveyed by events and/or corre-

sponding image or map pixels. Direct methods rely on the quasi-continuous nature of event data, for example to compute an incremental camera pose from the previously estimated one: the increment is small, as events are continuously triggered without gaps or blind times.

*event alignment*

Among direct methods, a prominent subclass —due to their state-of-the-art accuracy performance—is that of methods that estimate motion or scene parameters by event alignment, which appears in the form of sharp images of warped events (IWEs). The idea is to estimate motion by “undoing it”, *i.e.*, finding the parameters that motion-compensate the event data. Event alignment can be measured by means of different objectives: variance, gradient magnitude, dispsoner, etc. They are equivalently known as Focus or Contrast Maximization (CMax) [360, 1007]. In problems where events can be warped to a few pixels or a line, these methods can suffer from undesired global optima [1006]. Data association in direct methods is typically handled implicitly and in a soft manner, inherited by the distance in the pixel grid. Nevertheless, hard associations using nearest-neighbor values are also possible and effective in some cases.

*contrast maximization*

#### **10.4.4 Model-based and Learning-based Methods**

So far, the majority of event-based SLAM approaches are hand-crafted, designed by human intuition based on the principles of operation of the event camera and the SLAM problem. Instead, deep-learning methods leverage artificial neural networks (ANNs) to model event data, either by converting events into image-like representations or by processing them directly with Spiking Neural Networks (SNNs). These methods are often categorized into supervised or self-supervised, depending on the type of supervisory signal. Self-supervised methods rely on events or other sensors (*e.g.*, colocated grayscale images) to estimate depth and camera pose by leveraging some temporal dynamics or photometric consistency loss [1240]; whereas supervised methods require ground truth data for training [584], which is typically difficult to acquire in the real world. In recent years, many multi-modal datasets have been recorded onboard cars, drones, etc., which can provide the data needed for ANN training.

*generalization*

Learning-based solutions may substitute parts of the SLAM pipeline, such as feature extraction and tracking [757], or try to replace the entire system (end-to-end). Learning-based approaches offer the advantage of handling complex data representations and noise implicitly, but require large datasets for training and may suffer from generalization issues when applied to significantly different scenes (*i.e.*, “domain shift”) from the ones in the training set.

## 10.5 Back-end of an Event-based SLAM System

The goal of a refinement module like the SLAM back-end [136] is to improve the consistency between the variables of the SLAM problem and the sensor data, thus improving accuracy and robustness of the fit, reducing the propagation of errors between tracking and mapping modules of the system. As we discussed in earlier chapters, bundle adjustment (BA) is commonly used in visual SLAM back-ends.

Event-based BA is still in its infancy, as most event-based SLAM systems lack a refinement step. Instead, they operate in a parallel tracking-and-mapping manner [570, 911, 1299], with each module relying on the output of the other concurrently running module as input to work properly. Current event-based SLAM systems have prioritized simplicity and taking advantage of the low-latency benefits of event cameras over accuracy and robustness. In addition to the challenges mentioned in Section 10.2 (noise, motion-dependent appearance, etc.), an event-based back-end poses the challenge of jointly estimating correlated variables, which implies a high-dimensional search space, making optimization costly (in terms of complexity and latency) and prone to local minima.

Only recently BA has been used in systems that include event cameras. Since the back-end of a SLAM system is highly determined by the output of the front-end (as there needs to be a tight integration between both modules for best performance), we categorize event-based back-ends as indirect (feature-based) or direct (photometric-based).

**Indirect back-ends** are inherited from classical indirect frame-based methods [1104, 643, 784]. They operate on geometric primitives (corners, lines, etc.) that are detected in the event stream (possibly preceded by an events-to-image conversion [204, 941] to reutilize frame-based detectors). The objective typically consists in the minimization of the reprojection error, measured by the Euclidean distance in the image plane [437]. This approach has the advantage of reutilizing mature, robust techniques in classical SLAM. However, it discards the large amount of information contained in the events (as revealed by image reconstruction methods [915, 1277]) and often falls short of achieving the desired performance: due to noise and the dependency of events on motion, current event corners are not as accurate and stable as frame-based ones, hence their use in SLAM has been scarce [609]. Examples of indirect back-ends include [941, 204, 1151].

*bundle adjustment*

*joint estimation*

*image reconstruction*

**Direct back-ends** work on sensor data (rather than geometric primitives) and the objective typically consists in the minimization of some form of photometric error. Hence they are more tailored than indirect ones. Approaches like [461], which leverage grayscale information from colocated frames, borrow the back-end from frame-based systems [308, 26]. However, grayscale frames can suffer from motion blur and low dynamic range. Event-only back-ends do not suffer from these limitations; they are recent and so far have been developed for constrained motions (planar or rotational). The objective may consist in the maximization of event align-

ment (also called motion compensation or CMax) [359, 416] or the minimization of the photometric error (*i.e.*, temporal contrast) conveyed by each event [417, 418]. They are designed based on the event generation model (10.1). As each event carries little information and the number of problem unknowns in SLAM is typically large, many events are needed for accurate BA, which poses demands on computational resources, power and latency. There is plenty of room for investigation of efficient direct, event-only BA in natural scenes and 6-DoF motion scenarios.

### 10.6 State-of-the-Art Systems

Table 10.1 collects concrete systems in event-based VO/SLAM, describing some of their characteristics (direct, indirect, etc.) according to the categorization introduced in previous sections. While it is not possible to describe all of them in detail in this chapter (and neither is our intention), certain trends are worth mentioning.

The literature is dominated by model-based systems; data-driven approaches have not taken over yet (although that might happen in the near future, as it occurred with other computer vision tasks). Ever since the beginning, the problem of SLAM with event cameras has been tackled under different assumptions, increasing the complexity in terms of (i) camera motions, (ii) type of scenes, and (iii) additional sensors (or information, such as a map of the scene) to simplify the problem (*e.g.*, a depth sensor attached to an event camera decreases the burden of depth estimation from events alone, and IMUs provide accurate angular velocity information, etc.).

Once an event-based method shows good performance, it is incrementally improved in an almost standard “exploitation” roadmap (similar to frame-based SLAM): for example, monocular methods [911] can be extended into stereo or multi-camera scenarios [381], event-only methods like [1299] (resp. [584]) can be robustified using inertial data fusion [682, 809] (resp. [408]), base system can be extended to handle omnidirectional lenses, etc. Despite this “exploitation” path, event-based SLAM is still an emerging field and, therefore, is in an exploration phase (of different techniques). This becomes evident when analyzing the methods in Table 10.1: diverse ideas and principles, leading to different map representations, event representations, loss functions, etc., are leveraged to design the estimation algorithms underpinning these systems. There is still plenty of room to investigate new state estimation ideas, especially those that take advantage of the genuine characteristics of the sensor.

*exploration-exploitation*

### 10.7 Datasets, Simulators, and Benchmarks

Prototyping, training and benchmarking event-based vision systems places high demands for high-quality, diverse, and rich data (real and synthetic). The development of simulators, datasets, and leaderboards is essential to move the field forward and establish a common and solid ground in scientific and technical progress. Next we describe prominent SLAM datasets, benchmarks, and simulators for event cameras.

System	M/DL	I/D	Event represent.	BA	Motion	Scene	Input	Remarks
Cook [229]	M	D	Event Frame	✗	Rot	Natural	E	Interacting network using optical flow
Weikersdorfer [1172]	M	I	Individual Event	✗	Planar	2D B&W	E	First filter-based Ev-SLAM.
PF-SMT [569]	M	D	Individual Event	✗	Rot	Natural	E	Two interleaved Bayesian filters
Censi [160]	M	D	Event Packet	✗	6DoF	B&W	E+F+D	Filter-based VO based on image gradient
EB-SLAM-3D [1173]	M	D	Individual Event	✗	6DoF	Natural	E+D	Augment events with depth sensor
Yuan [1257]	M	I	Event Frame	✗	6DoF	B&W	E+I+M	Vertical line-based camera tracking
Kueng [609]	M	I	Local Point Set	✗	6DoF	Natural	E+F	Event-based feature tracking VO
ETAM [570]	M	D	Individual Events	✗	6DoF	Natural	E	Three interleaved filters
CMax - $\omega$ [356]	M	D	Individual Events	✗	Rot	Natural	E	Contrast Maximization
EVO [911]	M	D	Edge Map	✗	6DoF	Natural	E	Event-event geometric alignment
EVIO [1300]	M	I	Point sets	✗	6DoF	Natural	E+I	Filter-based and MC features
Rebecq [912]	M	I	MC Event Images	✓	6DoF	Natural	E+I	Feature-based, sliding-window back-end
RTPT [922]	M	D	Individual Events	✗	Rot	Natural	E	Panoramic tracker and mapper
Gallego [358]	M	D	Individual Events	✗	6DoF	Natural	E+M	Resilient sensor model
Mueggler [776]	M	D	Individual Events	✓	6DoF	Natural	E+I+M	Continuous-time pose estimator
USLAM [941]	M	I	MC Event Images	✓	6DoF	Natural	E+F+I	Sensor fusion & sliding-window back-end
Chin [204]	M	I	Event Frames	✓	Rot	Stars	E	Tailored to star tracking
ESVO [1299]	M	D	Time surfaces (TS)	✗	6DoF	Natural	2E	Stereo matching on TS patches
Hadjiger [425]	M	I	Corners on TS	✗	6DoF	Natural	2E	Cross-corr. feature descriptors
CMax-GAE [571]	M	D	Individual Events	✗	Rot	Natural	E	Contrast maximization
EKLT-VIO [722]	M	I	Individual events	✓	6DoF	Natural	E+F+I	EKLT tracker and VIO back-end
EDS [461]	M	D	Event images	✓	6DoF	Natural	E+F	Frame-based back-end (DSO)
CB-VIO [681]	M	I	Individual events	✓	6DoF	Natural	E+F+I	Feature tracker and VIO back-end
Wang [1151]	M	I	Binary images	✓	6DoF	Natural	2E	Feature matching
El Moudni [303]	M	D	Time Surfaces TS	✗	6DoF	Natural	2E	Use ESVO tracker and EMVS mapper
ESVIO [183]	M	I	Time surfaces (TS)	✓	6DoF	Natural	2E+2F+I	Feature tracking on from TS
ESVIO-direct [682]	M	D	Time surfaces (TS)	✓	6DoF	Natural	2E+I	Extension of ESVO
PL-EVIO [410]	M	I	Time surfaces (TS)	✓	6DoF	Natural	E+F+I	Point & line features, sliding-window BA
CMax-SLAM [416]	M	D	Individual Events	✓	Rot	Natural	E	Contrast Maximization refines motion
EVI-SAM [409]	M	D,I	Individual Events	✓	6DoF	Natural	E+F+I	Dense mapping
Zuo [1314]	M	D	Individual Event	✗	6DoF	Natural	E+D	Augment events with depth sensor
DEVO [584]	DL	I	Event voxel grids	✓	6DoF	Natural	E	Event-version of DPVO [1081]
EMBA [417]	M	D	Individual Events	✓	Rot	Natural	E	Refines motion and gradient map
EPBA [418]	M	D	Individual Events	✓	Rot	Natural	E	Refines motion and intensity map
ES-PTAM [381]	M	D	Events (also as frames)	✗	6DoF	Natural	2E	Use EVO tracker and EMVS mapper
ESVO2 [809]	M	D	Time surfaces (TS)	✓	6DoF	Natural	2E+I	Extension of ESVO
DEIO [408]	DL	I	Event voxel grids	✓	6DoF	Natural	E+I	Extension of DEVO and DPVO

Table 10.1 *Summary of Event-based Visual SLAM methods, sorted chronologically. The columns indicate: the type of method (Model-based or Deep-Learning-based, Direct or Indirect), whether the method has a global refinement module (i.e., back-end / BA), the type of camera motions (Rotational, Planar, 6-DoF) and scenes (high-contrast black-and-white, etc.) it can handle, and the type of input data used (Event camera, Frame-based camera, Depth sensor, IMU and Map), where “2E” means stereo events (two sensors).*

### 10.7.1 Simulators

There are a variety of publicly available tools for generating high-quality synthetic event camera data. ESIM [914] is an evolved version of [775], which was one of the first simulators to mimic the principle of operation of an event camera. Previous efforts, such as [534], just thresholded the difference between two successive frames to create edge-like images that resembled the output of an event camera. ESIM tightly couples the rendering engine and the event simulator, which allows the latter to adaptively render frames based on the dynamics of the visual signal.

The event camera simulator in CARLA [279] expands ESIM in more diverse, rich,

adaptive rendering

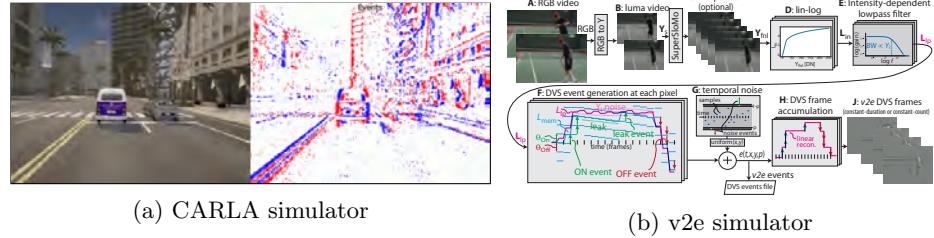


Figure 10.5 Event camera simulators: (a) RGB image and generated events using the ESIM simulator in CARLA [279]; (b) Detailed data processing steps of the V2E tool [481].

and complex scenarios for autonomous driving. In the context of learning monocular depth from events [460], the event camera sensor developed in CARLA takes the rendered images from the simulator and computes per-pixel brightness changes to simulate an event camera in the same way as in ESIM. Figure 10.5 shows RGB images and events generated in the CARLA simulator.

Motivated by learning-based approaches that require large amounts of event data for training and the fact that event data are hardly available due to the novelty of the sensor, a tool for converting any existing video recorded with a conventional camera into synthetic event data was developed: Video to Events (Vid2E) [371]. Hence, Vid2E aims at reducing the gap between publicly available datasets in traditional and event-based computer vision by enabling the use of a virtually unlimited number of existing video datasets for training networks designed for real event data. Vid2E solves the event data scarcity bottleneck by combining ESIM and adaptive video frame interpolation. ESIM can address this problem by adaptively rendering *virtual* scenes at arbitrary temporal resolution. However, video sequences typically only provide intensity measurements at fixed and low temporal resolution. Super SloMo [520] allows to reconstruct frames at arbitrary temporal resolution and then applies the event camera simulator ESIM. The number of intermediate frames is carefully chosen since, on the one hand, a low value leads to aliasing of the brightness signal, and on the other hand, high values impose a computational burden.

#### *noise model*

An important aspect of an event camera simulator is to accurately model noise, to reduce the simulation-to-real gap when transferring the algorithms. For example, ESIM, and therefore its derivative simulators, implement a simple noise model based on empirical observation [659]: the contrast threshold of an event camera ( $C$  in (10.1)) is not fixed but follows a normal distribution. To simulate this, at each step of the simulation, ESIM samples from a normal distribution  $\mathcal{N}(C, \sigma_C^2)$ , where the noise level  $\sigma_C$  can be adjusted. Additionally, ESIM allows for separate positive and negative contrast thresholds ( $C^+$  and  $C^-$ ) to more accurately simulate a real event camera. Other noise effects, such as spatial and temporal variations in contrast thresholds due to electronic noise or the limited bandwidth of event pixels, shall also be considered in event camera simulators.

Vid2E [371] models an ideal event camera lighting. V2E [481] proposes instead a more realistic noise simulator of an event camera based on the DVS circuitry. V2E is the first event camera simulator that includes temporal noise, leak events, and finite intensity-dependent bandwidth, including the same Gaussian threshold distribution as in Vid2E. V2E is a step forward towards a more realistic simulator enabling the generation of synthetic datasets covering a range of illumination conditions, which is an important use case for events. Similarly to Vid2E, V2E uses Super SloMo [520] to increase the temporal resolution of the input video. Figure 10.5 depicts the V2E architecture in detail.

Simulating event camera noises is a challenging topic for realistic synthetic event generation, EventGan [1303] proposes an end-to-end approach using deep learning to simulate event camera data. Their work proposes a method that leverages the existing labeled data for images by simulating events from a pair of temporal image frames, using a U-Net [931] encoder-decoder network. The methodology consists of training a neural network on pairs of images and events. Instead of applying a direct numerical error loss, they use an adversarial discriminator loss and a pair of cycle consistency losses. EventGAN generates a 3D spatio-temporal voxel grid for each polarity (instead of a set of individual events). This voxel grid representation is commonly used as input to ANNs.

VISTA 2.0 [30] is a simulator that integrates multiple sensor types, including RGB cameras, LiDAR, and event-based cameras, to facilitate policy learning for autonomous vehicles. It uses high-fidelity, real-world data to simulate diverse scenarios, such as varying weather, lighting, and road conditions. The event camera simulator works similarly to ESIM with adaptive sampling. The bidirectional optical flow between two consecutive frames is estimated using an ANN. VISTA 2.0 is designed for training perception-to-control policies, demonstrating enhanced robustness and sim-to-real transfer capabilities compared to real-world training data alone, thereby improving vehicle control in safety-critical situations.

Video to Continuous Events (V2CE) [1279] tackles the problem of producing events with more realistic timestamps than previous simulators. Vid2E and V2E generate events at discrete timestamps, instead of a continuous-time fashion like real events. This is not negligible in tasks that are sensitive to timestamp distribution, which prohibits the use of synthetic events since they can bring a significant domain shift with respect to real events. V2CE simulator works in two stages. The first stage consists of a supervised 3D U-Net encoder-decoder ANN that predicts two voxel grids (one per polarity, similar to EventGAN) from video. The second stage recovers precise event timestamps from the voxel grids. The method iteratively deduces the event count and their relative positions in each voxel. V2CE also shows that it can accurately generate events in saturated light areas and in edges where the event generation model for an ideal sensor does not hold.

### **10.7.2 Datasets and Benchmarks**

*motion-capture system*

The number of event-based datasets dedicated to Visual Odometry and SLAM has increased significantly since the publication of the ECDS [775] (see Table 10.2). ECDS was the first dataset with synchronized events, IMU, and ground-truth camera poses in 6-DoF. Previous datasets [952] included both synthetic and real events featuring pure rotational motion (3 DoF) in simple scenes with high visual contrast; ground-truth data was obtained using an IMU. Other work [55] enabled a 5 DoF comparison of event-based and frame-based camera movements; and ground truth was obtained from the pan-tilt unit encoders and the ground robot’s wheel odometry, making it prone to drift. ECDS contains hand-held, 6-DoF motion (slow- and high-speed) on a variety of scenes with precise ground-truth camera poses from a motion-capture system. The dataset consists of 11 scenes with real events and two additional scenes with synthetic events. The synthetic data was produced with the first version of what became the ESIM [914] simulator.

The RPG stereo dataset [1298] consists of eight hand-held sequences recorded with a stereo DAVIS [111] in an office environment and a synthetic sequence (featuring three fronto-parallel planes at various depths) produced by the simulator [775]. Although this dataset does not provide ground-truth depth, it has accurate ground-truth poses from a motion capture system and serves as a good starting point for prototyping and evaluating event-based stereo SLAM methods.

The Multi Vehicle Stereo Event Camera Dataset (MVSEC) [1301] is the first dataset to offer ground-truth depth across a variety of platforms. It captures both indoor and outdoor scenes with varying levels of illumination and movement speeds. The platforms include a handheld rig, a hexacopter, a car, and a motorcycle, all equipped with calibrated sensors like 3D LiDAR, IMUs, and standard frame-based cameras. MVSEC has wide-ranging applications in pose estimation, mapping, obstacle avoidance, and 3D reconstruction, offering accurate ground-truth depth and pose data through its integrated LiDAR system. The datasets contain long sequences that enable a comprehensive evaluation of event-based algorithms.

*segmentation*

The UZH-FPV (First Person View) dataset [257] is specifically designed to advance research in autonomous drone racing. It features a custom-built quadrotor with a Qualcomm Flight Board and an mDAVIS346 [1071] event camera mounted on a Lumenier QAV-R carbon fiber frame. The recordings capture indoor and outdoor scenes at varying speeds and trajectories, which present challenges for navigation and state estimation. This dataset supports research in VIO, event data processing, and real-time drone applications in fast scenarios. It has become a key resource for developing high-speed camera motion algorithms, particularly in autonomous drone racing, and has also been used for competitions at conferences and workshops.

The Event Camera Motion Segmentation Dataset (EV-IMO) [766] is the first event-based dataset created specifically for segmentation of independently moving objects (IMO) in indoor environments. It contains 32 minutes of recordings, track-

Dataset	Platforms	Pixel Resolution	Sensors
ECDS [775]	Hand-held	240 × 180	E, F, I
RPG-stereo [1298]	Hand-held	240 × 180	2E
MVSEC [1301]	Hand-held, Drone, Car, Byke	346 × 240	2E, 2F, I, Lidar, GPS
UZH-FPV [257]	Drone	346 × 260	E, F, I
EV-IMO [766]	Hand-held	346 × 260	E, F, I, Depth
EV-IMO2 [126]	Hand-held	640 × 480	3E, F, I, Depth
DSEC [372]	Car	640 × 480	2E, 2F, Lidar, GPS
TUM-VIE [583]	Hand-held	1280 × 720	2E, 2F, I
EDS [461]	Hand-held	640 × 480	E, F(RGB), I
VECTor [363]	Hand-held	640 × 480	2E, 2F, RGB-D, I, Lidar
M2DGR [1247]	Ground Robot	640 × 480	E, F, I, Lidar, GPS, Thermal
ViViD++ [633]	Hand-held, Car	240 × 180, 640 × 480	E, F, RGB-D, Thermal, Lidar, GPS
FusionPortable [522]	Hand-held, Quadruped Robot	346 × 240	2E, 2F, I, Lidar, GPS
Stereo HKU-VIO [183]	Hand-held	346 × 260	2E, 2F, I
M3DE [163]	Drone, Car, Quadruped Robot	1280 × 720	2E, 2F, I, Lidar, GPS
CoSEC [858]	Car	1280 × 720	2E, 2F, I, Lidar, GPS

Table 10.2 *Summary of event-based SLAM datasets, sorted chronologically. Same notation for sensor data as in Tab. 10.1. Stereo and multi-sensor datasets are further described in the survey [380].*

ing up to three fast-moving IMOs using a motion capture system. The dataset provides pixel-wise motion masks, and ground-truth egomotion and depth. It is useful in robotics, especially in scene-constrained environments where accurate motion detection is crucial for tasks like object tracking and autonomous navigation. EV-IMO2 [126] builds on its predecessor by offering more sequences, three higher quality event cameras, and more complex scenarios. This version serves as both a challenging benchmark for current algorithms and a rich training set for developing new methods, including event-based SLAM in monocular and stereo setups.

The Stereo Event Camera Dataset for Driving Scenarios (DSEC) [372] is large-scale, intended to support research in autonomous driving, especially in developing robust perception systems capable of handling adverse lighting conditions through sensor fusion of events and frames. The dataset features a platform with a multi-camera setup, including two VGA-resolution event cameras (Prophesee Gen 3.1) with a 60 cm baseline and two RGB cameras (FLIR Blackfly S) with a 51 cm baseline (see Fig. 10.6). The setup includes a Velodyne VLP-16 LiDAR and an RTK GPS for precise localization. Data were collected in various urban and rural settings in Switzerland under diverse illumination conditions, such as day, night, and direct sunlight, providing ground-truth depth maps for stereo matching. DSEC also provides Optical Flow and Disparity to benchmark algorithms in challenging driving conditions. These benchmarks use metrics like N-pixel disparity error, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) to assess the performance of algorithms combining high-resolution event camera data with RGB frames.

The TUM Stereo Visual-Inertial Event Dataset (TUM-VIE) [583] employs stereo Prophesee Gen4 event cameras (1 Megapixel resolution) along with synchronized

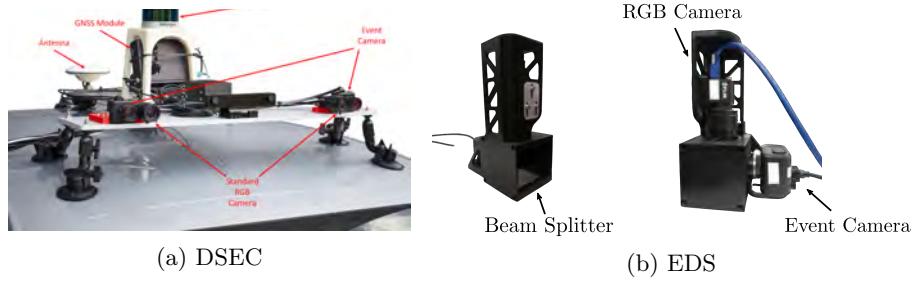


Figure 10.6 Details of some Event-SLAM datasets: (a) the sensor suite mounted on top of a car, in the DSEC [372] dataset (©2021 IEEE). (b) Details of the beamsplitter that allows the sensors to share a spatially aligned field of view in the EDS [461] dataset (©2022 IEEE).

IMU data at 200Hz and stereo grayscale frames at 20Hz. It includes sequences from handheld and head-mounted setups in diverse indoor and outdoor environments, covering various scenes such as sports activities, HDR scenarios, and low-light conditions. TUM-VIE is intended to facilitate research on VIO, SLAM, 3D reconstruction, and sensor fusion, especially in challenging conditions where traditional methods may fail, pushing the boundaries of high-resolution event-based perception algorithms.

The Event-Aided Direct Sparse Odometry (EDS) dataset [461] includes high-quality events, color frames, and IMU data to support research in monocular VIO. Data were acquired using a custom-made beamsplitter device (see Fig. 10.6), allowing for precise alignment of RGB frames and events on the same optical axis, which is not commonly found in previous datasets. The scenes recorded include natural indoor environments, providing high-resolution, well-calibrated data for applications like optical flow estimation, depth estimation, and robust visual odometry under various motion and lighting conditions.

The Versatile Event-Centric (VECtor) Benchmark Dataset [363] is also designed to evaluate event-based SLAM algorithms. The recording platform holds a diverse sensor suite, including stereo cameras (event- and frame-based), an RGB-D sensor, a 128-channel LiDAR, and a nine-axis IMU, all mounted on a versatile 3D-printed holder. The dataset features both small-scale indoor environments, like a motion capture arena, and large-scale indoor environments with various complexities and illumination conditions. It claims to offer high-resolution (VGA), synchronized data across diverse environments, ensuring reliable evaluation of SLAM algorithms in both static and dynamic, low-light, and HDR scenarios. This makes it a comprehensive resource for advancing research in multi-sensor SLAM applications.

The Vision for Visibility Dataset (ViViD++) [633] was recorded with a multi-sensor platform, including thermal cameras, to support research on SLAM algorithms that can handle poor visibility, motion disturbances, and appearance changes, leveraging the complementary strengths of different sensors. The Fusion-

Portable dataset [522] includes a Quadruped robot that moves in various scenes, such as corridors, canteens, roads, and gardens under different lighting conditions.

Finally, the Multi-robot, Multi-Sensor, Multi-Environment Event Dataset (M3ED) [163] (informally known as MVSEC 2.0) is focused on high-speed dynamic motions in robotics applications. It combines 1 Megapixel stereo event cameras, grayscale and RGB cameras, a 64-beam LiDAR, and high-quality IMU, all synchronized with RTK localization. Unlike previous datasets, M3ED offers heterogeneous data from multiple platforms in both structured and unstructured environments, with ground truth pose, depth, and semantic labels, making it a valuable resource for developing robust event-based perception algorithms for dynamic environments beyond traditional driving or indoor applications.

### 10.7.3 Metrics

Ideally, SLAM systems should characterize the quality of their localization and mapping modules individually. However, because (i) both modules operate in an intertwined way (depth errors affect camera pose errors, and vice versa), and (ii) ground-truth localization information is considerably more compact (6-DoF) and easier to acquire than accurate ground-truth depth, the result is that depth estimation errors are often *subsumed* in the evaluation of camera trajectory errors.

Conceptually, since both classical SLAM and event-based SLAM output camera trajectories, event-based SLAM inherits the performance evaluation protocol from classical SLAM. Two commonly used metrics are the Absolute Trajectory Error (ATE) and the Relative Pose Error (RPE) [1042]. The ATE assesses the accuracy of the camera's pose relative to a fixed world coordinate system; hence, it provides a broad assessment of the VO system's long-term performance. The RPE evaluates the relative poses between consecutive (*i.e.*, nearby) timesteps; hence, it focuses on the local consistency of VO system. The translational error in ATE, also known as positional error, is calculated as the Euclidean distance between the estimated and ground-truth camera positions. The rotational error, or orientation error, is determined by the geodesic distance in  $SO(3)$ . Similarly, the translational and rotational parts of RPE are calculated between pairs of camera poses over a time interval. Some studies also compute the positional error relative to the mean scene depth or total distance traveled, ensuring that the error remains invariant to the scale of the scene or trajectory.

Additional error metrics —Average RPE (ARPE), Average Relative Rotation Error (ARRE) and Average Endpoint Error (AEE)— may be used to assess the estimated translation vectors and rotation matrices [1302]. Specifically, ARPE and AEE measure differences in position and orientation between two translation vectors, while ARRE calculates the geodesic distance between two rotation matrices.

Beyond these metrics, average linear and angular velocity errors can also be useful for evaluating camera pose estimation, especially when working with event

*geodesic distance*

cameras, where abrupt and fast camera motions are estimated thanks to the events. Camera poses are functions of both velocities over time. Several toolboxes [405, 1280] are publicly available to facilitate the reproducibility of research and reduce the complication in SLAM trajectory evaluation.

In case depth estimation is evaluated separately, the average depth error at various cutoffs up to fixed depth values is often used, allowing for comparison across methods on different scales of 3D maps. However, there are not many datasets that contain ground-truth depth information (see Sec. 10.7). The Root Mean Square Error (RMSE) of the Euclidean distance between the estimated 3D point with respect to the closest surface on the ground truth map is the preferred metric. Additional metrics, such as the Relative Error (REL) and completion (number of points recovered), are also used in the literature [302, 460, 381].

completion

neuromorphic computing

co-design

interdisciplinar

sensor fusion

## 10.8 Further Readings & Recent Trends

Although research on event-based SLAM has made considerable progress, many open questions and problems remain given the novelty of the technology. These questions pertain to what are the best ways (hardware and software) to acquire and process visual information for a given task (*e.g.*, SLAM) in order to rival or surpass (in terms of robustness, latency, efficiency, accuracy, etc.) the performance observed in biological species.

The sensor is asynchronous , but most of the systems in the literature are designed on serial (*i.e.*, von Neumann) processors (due to the entry barrier to neuromorphic computing). This is suboptimal in terms of efficiency (power consumption), latency, etc. compared to the expected performance of fully neuromorphic systems [842], where event cameras are paired with asynchronous (brain-inspired, spike-based) processors, controllers, actuators, etc. It is a long-standing dream of the research community: to build robots that mimic the efficient processing of animals and their ability to map and localize themselves in the environment (with potential applications in “always on” inside-out tracking for AR/VR, etc.). This dream requires rethinking and co-designing sensors, processors, and algorithms [245] in a neuromorphic engineering way, which is very challenging, as it takes great breadth and depth of expertise, and coordination of multiple disciplines.

In the near future, novel hybrid sensors are being developed that provide data inspired by the two visual streams [1235], spatially and temporally aligned, with low latency, HDR, and fine details (pixel count). Alternatively, foveated sensors [317], mimicking biology, are also investigated to reduce bandwidth requirements. There is still a big field to explore in terms of event cameras, their evolution (*e.g.*, near-sensor processors like pixel processor arrays, Aeveon sensors), and their combination with other sensors (frame-based cameras, structured light, LiDAR, radar, etc.) for data fusion and improved SLAM performance.

**Acknowledgments**

The authors thank Giovanni Cioffi for his support in preparing this chapter.

# 11

## Inertial Odometry for SLAM

Guoquan (Paul) Huang, Cédric Le Gentil, Teresa Vidal-Calleja,  
Davide Scaramuzza, Frank Dellaert, and Luca Carlone

Inertial Measurement Units (IMUs) have become one of the most pervasive sources of odometry for robot simultaneous localization and mapping. An IMU measures the linear acceleration and the rotation rate of the body the sensor is attached to. IMUs are available in a broad range of form factors, costs, and performance levels, from large and accurate optical sensors used on airplanes to small but more noisy micro-electromechanical systems (MEMS) used in smart phones and other consumer devices. The low-SWAP and inexpensive nature of MEMS IMU sensors renders them great candidates as sensors for robotics, where these sensors have been extensively studied with application to SLAM for more than two decades.

In this chapter, we first introduce basic facts about IMUs and describe their measurement model (Section 11.1). Then, we introduce the concept of IMU preintegration (Section 11.2), which allows adding high-rate IMU data into a factor graph optimization framework. Next, we observe that using IMU data introduces extra variables in the optimization (*e.g.*, sensor biases) and discuss observability<sup>1</sup> properties of systems that combine IMUs with exteroceptive sensors, *e.g.*, camera or LiDAR (Section 11.3). Finally, we showcase examples of what's achievable with modern IMU-centric SLAM systems (Section 11.4) and review recent trends (Section 11.5).

### 11.1 Basics of Inertial Sensing and Navigation

*accelerometer*

*gyroscope*

A 6-axis Inertial Measurement Unit (IMU) comprises an *accelerometer*, which measures the linear acceleration of the sensor with respect to an inertial frame, and a *gyroscope*, which measures the angular velocity (or rotation rate) of the sensor.<sup>2</sup> Traditionally studied in aerospace engineering, *inertial navigation systems* (INS) aim at estimating the current state (*e.g.*, pose, velocity) of the platform the

<sup>1</sup> Observability establishes under which conditions the estimation problem is well posed, *i.e.*, whether it is at all possible to compute an estimate close to the ground truth given the measurements.

<sup>2</sup> An IMU typically also includes a *compass* that measures the direction to the magnetic north. This sensor is less used in SLAM applications since in many robotics applications, including in indoor and urban environments, it exhibits large biases. These biases are induced by local magnetic disturbances, which can be caused by, *e.g.*, large metallic structures and electronic devices.

IMU is mounted on, from the initial state and the history of the IMU measurements [171, 1096]. Different INS can be categorized into *strapdown systems*, where the IMU is mounted to the frame of the platform, and *stabilized systems*, where the IMU is mounted on an inner gimbal, multi-gimbal structure, or floating ball, which is designed to maintain its orientation constant with respect to an inertial frame. Most INS in robotics fall into the former category, *i.e.*, they rely on an IMU that measures the local linear acceleration and angular velocity of the sensing platform it is rigidly connected to. In robotics, the term *inertial odometry* is commonly used as a synonym of inertial navigation, to emphasize the odometric nature of the estimate.

Clearly, the odometric estimate produced by an INS drifts over time, so in most applications the estimation also relies on other sensors (*e.g.*, GPS, cameras, LiDARs), in which case one talks about *aided inertial navigation systems* (AINS). In robotics, it is common to directly specify the combination of sensors used with the IMU. For instance, a system that combines cameras and IMUs to provide 3D motion tracking is called a *visual-inertial odometry*, while a system that also includes loop closures is called a *visual-inertial SLAM* system.

*visual-inertial SLAM*

### 11.1.1 Sensing Principles and Measurement Models

An IMU commonly includes a 3-axis accelerometer and a 3-axis gyroscope, measuring the angular rate and the linear acceleration of the sensor platform. The basic principle underlying gyroscope design is the conservation of angular momentum. On the other hand, an accelerometer uses the inertia of a mass to measure the difference between the kinematic acceleration with respect to the inertial frame and the gravitational acceleration. Different principles can be used for the design of accelerometers, for example, by using a rate gyroscope mounted as a pendulum mass, based on the inertia of a proof mass inside a low-friction case, or based on the difference in vibration of two thin metal tapes suspended inside a case with a proof mass suspended between them.

**Measurement Model.** We now describe the IMU measurement model, which relates the IMU measurements to the state of the robot and other quantities (*e.g.*, biases) we need to estimate. For simplicity, we assume the sensor frame coincides with the body frame  $\mathcal{F}^b$  of the robot, and the world frame  $\mathcal{F}^w$  is an inertial frame.<sup>3</sup> The IMU measurements collected at time  $t$ , namely  $\mathbf{a}(t)$  and  $\boldsymbol{\omega}(t)$ , are typically assumed to be corrupted by additive white Gaussian noise  $\boldsymbol{\eta}$  and slowly varying

*inertial frame*

<sup>3</sup> In aerospace, it is standard practice to distinguish non-inertial navigation frames, *e.g.*, Earth-Centered Earth-Fixed (ECEF) and Local Geodetic Vertical (LGV) frames, from inertial ones, *e.g.*, Earth-Centered Inertial (ECI) [318]. In robotics, this distinction is often de-emphasized in near-Earth small-scale applications where noisy low-cost IMUs are often used, as the impact of the Earth rotation is negligible compared to the measurement noise. For this reason, the world frame  $\mathcal{F}^w$ , which in robotics is typically chosen to be fixed at a location on Earth, is approximately treated as an inertial frame.

sensor biases  $\mathbf{b}$ :

$$\mathbf{a}(t) = \mathbf{R}_w^b(t)(\mathbf{a}^w(t) - \mathbf{g}^w) + \mathbf{b}^a(t) + \boldsymbol{\eta}^a(t), \quad (11.1)$$

$$\boldsymbol{\omega}(t) = \boldsymbol{\omega}_b^b(t) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t). \quad (11.2)$$

As usual, the superscript  $\mathcal{F}^b$  denotes that the corresponding quantity is expressed in the body (IMU) frame  $\mathcal{F}^b$ . The pose of the IMU at time  $t$  is described by the transformation  $\{\mathbf{R}_b^w(t), \mathbf{p}^w(t)\}$ , which maps a point from sensor frame  $\mathcal{F}^b$  to  $\mathcal{F}^w$  (note that  $\mathbf{R}_w^b(t) = (\mathbf{R}_b^w(t))^\top$ );  $\mathbf{a}^w(t) \in \mathbb{R}^3$  is the acceleration of the sensor in the world frame;  $\mathbf{g}^w$  is the gravity vector in the world frame. Therefore, the term  $\mathbf{R}_w^b(t)(\mathbf{a}^w(t) - \mathbf{g}^w)$  is the acceleration experienced by the IMU in the Body/IMU frame. The vector  $\boldsymbol{\omega}_b^b(t) \in \mathbb{R}^3$  is the instantaneous angular velocity of  $\mathcal{F}^b$  relative to  $\mathcal{F}^w$  expressed in coordinate frame  $\mathcal{F}^b$ . The noise terms  $\boldsymbol{\eta}^g(t)$  and  $\boldsymbol{\eta}^a(t)$  are assumed to be zero-mean Gaussian random variables, and the to-be-estimated biases  $\mathbf{b}^a(t)$  and  $\mathbf{b}^g(t)$  are assumed to follow random walks. Note that here the superscripts for noise and bias vectors do not refer to the frames but the sensors (accelerometer and gyroscope); *e.g.*,  $\mathbf{b}^a(t)$  is the accelerometer bias.

**Extended Models.** While the IMU measurement model (11.1)-(11.2) often suffices in robotics, more sophisticated models that more accurately capture the sensing process may be needed, for example, when (re-)calibrating the sensor platform. Due to the imperfection in manufacturing, accelerometers may suffer from misalignment and scale errors, and the model (11.2) can be extended to:

$$\mathbf{a}(t) = \mathbf{T}_a \mathbf{R}_w^b(t)(\mathbf{a}^w(t) - \mathbf{g}^w) + \mathbf{b}^a(t) + \boldsymbol{\eta}^a(t), \quad (11.3)$$

where  $\mathbf{T}_a$  is the shape matrix that models both misalignment and scale errors in the accelerometer measurements. Scale errors can be made of static or temperature-related components and can be determined during sensor (intrinsic) calibration. Similarly, the gyroscope measurement model can be extended to capture misalignment and scale errors:

$$\boldsymbol{\omega}(t) = \mathbf{T}_g \boldsymbol{\omega}_b^b(t) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t), \quad (11.4)$$

*g-sensitivity*

where  $\mathbf{T}_g$  is the shape matrix that models both misalignment and scale errors in the gyroscope measurements. Gyroscope measurements are often influenced by acceleration, a phenomenon called *g-sensitivity*. The magnitude of this influence is considered negligible if it is within the range of the additive white noise  $\boldsymbol{\eta}^g(t)$ , while in some MEMS hardware, it can be more significant and modeled as follows:

$$\boldsymbol{\omega}(t) = \mathbf{T}_g \boldsymbol{\omega}_b^b(t) + \mathbf{T}_s \mathbf{R}_w^b(t)(\mathbf{a}^w(t) - \mathbf{g}^w) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t), \quad (11.5)$$

where  $\mathbf{T}_s$  is the g-sensitivity matrix, which can be estimated during calibration.

### 11.1.2 Initial Alignment

In SLAM it is customary to set the global coordinate frame to be the starting pose of the trajectory, *i.e.*, set the initial pose  $\{\mathbf{R}_b^w(0), \mathbf{p}^w(0)\}$  to be the identity pose. However, in INS, as the IMU measurements involve the gravitational force (*c.f.* (11.1)), we typically choose the world frame to be gravity-aligned, thus requiring to align the initial pose with the gravity direction. In other words, since the IMU measurements depend on the gravity direction, the orientation of the robot is no longer an arbitrary choice, and it must be consistent with the gravity direction. Specifically, we need to compute the rotation  $\mathbf{R}_b^w(0)$  that aligns the body (IMU) frame to the world frame. For simplicity, assume the robot is initially static, *i.e.*, at the beginning of deployment, no specific force is applied to the robot and the commonly-used low-cost MEMS IMU only measures the gravitational force. Clearly, given only the local gravity measurement  $\mathbf{g}^b$ , we cannot recover the rotation along gravity (*i.e.*, yaw), which is thus up to free choice depending on the application. However, we can determine the rotation corresponding to roll and pitch via the following static initialization:

$$\begin{cases} \mathbf{z}_w^b = \frac{\mathbf{g}^b}{\|\mathbf{g}^b\|} \\ \mathbf{x}_w^b = \frac{\mathbf{e}_1 - \mathbf{z}_w^b \mathbf{e}_1^\top \mathbf{z}_w^b}{\|\mathbf{e}_1 - \mathbf{z}_w^b \mathbf{e}_1^\top \mathbf{z}_w^b\|} \\ \mathbf{y}_w^b = \mathbf{z}_w^b \times \mathbf{x}_w^b \end{cases} \Rightarrow \mathbf{R}_w^b = [\mathbf{x}_w^b \quad \mathbf{y}_w^b \quad \mathbf{z}_w^b] \quad (11.6)$$

where we perform the Gram–Schmidt orthonormalization given vectors  $\mathbf{e}_1 = [1 \ 0 \ 0]^\top$  and  $\mathbf{g}^b$ , and  $\times$  is the cross product. Intuitively, the last column of the rotation matrix  $\mathbf{R}_w^b$ , namely  $\mathbf{z}_w^b$ , is the direction of the  $z$ -axis of the world frame with respect to the body frame. Since the  $z$ -axis of the world frame is aligned with gravity, eq. (11.6) computes  $\mathbf{z}_w^b$  from the measurement of the body-frame gravity vector  $\mathbf{g}^b$ . Then, the orthonormalization procedure computes orthonormal vectors  $\mathbf{x}_w^b$  and  $\mathbf{y}_w^b$  to complete the columns of the rotation matrix  $\mathbf{R}_w^b$  for an arbitrary choice of yaw.

**Alignment with High-end IMUs.** When using high-end IMUs, the gyroscope is sensitive enough to measure the Earth rotation rate  $\boldsymbol{\omega}_{ie}$ . In this case, assuming the chosen world frame is an inertial frame (e.g., the Earth-Centered Inertial frame, or ECI [318]), one can use the measurement of the body-frame gravity vector  $\mathbf{g}^b$  and the Earth rotation rate  $\boldsymbol{\omega}_{ie}$  to perform analytical alignment:

$$\begin{cases} \mathbf{g}^b = \mathbf{R}_w^b \mathbf{g}^w \\ \boldsymbol{\omega}_{ie}^b = \mathbf{R}_w^b \boldsymbol{\omega}_{ie}^w \\ \mathbf{g}^b \times \boldsymbol{\omega}_{ie}^b = \mathbf{R}_w^b (\mathbf{g}^w \times \boldsymbol{\omega}_{ie}^w) \end{cases} \Rightarrow \mathbf{R}_b^w = \left[ \begin{array}{c} \mathbf{g}^{w\top} \\ \boldsymbol{\omega}_{ie}^{w\top} \\ (\mathbf{g}^w \times \boldsymbol{\omega}_{ie}^w)^\top \end{array} \right]^{-1} \left[ \begin{array}{c} \mathbf{g}^{b\top} \\ \boldsymbol{\omega}_{ie}^{b\top} \\ (\mathbf{g}^b \times \boldsymbol{\omega}_{ie}^b)^\top \end{array} \right] \quad (11.7)$$

where the resulting rotation matrix  $\mathbf{R}_b^w$  typically needs to be projected onto  $\text{SO}(3)$  to mitigate the impact of measurement noise.

## 11.2 IMU Preintegration and Factor Graphs

In the previous section, we have introduced the IMU measurement model (11.1)-(11.2), which relates the IMU measurements to the state of the robot, and in particular its pose and velocity, as well as the sensor biases. While in principle we can use these models to derive a Maximum a Posteriori estimator as described in Chapter 1, this leads to impractically large factor graphs: a typical IMU provides measurements at a high-rate (*e.g.*, 200-1000 Hz) and the measurement model would require adding states to the factor graph at each IMU sampling time. The resulting factor graph would quickly become impractical to solve. The more astute reader might observe that a continuous-time formulation of the problem could circumvent the high-rate addition of variables to the factor graph. However, in a continuous-time formulation of inertial navigation, one would still need to add *factors* at high-rate, one for each measurement, again leading to an unwieldy factor graph.

*IMU preintegration*

In this chapter, we present the key idea of *IMU preintegration*, which provides a way to avoid adding states or measurements at IMU rate to the factor graph. The basic idea is to integrate IMU measurements over time to obtain relative motion measurements, such that these (fewer) motion measurements can be added to the factor graph instead. However, a naive integration of the IMU measurements (reviewed in Section 11.2.1) would still require repeating the integration of the measurements at each iteration of the factor graph solver (due to potential changes in the initial conditions for integration). *IMU preintegration* avoids this issue by separating terms that depend on the state variables from the measurements. The original idea of preintegration goes back to [706] and has been extended to operate on manifold in [334, 335]; in Section 11.2.2, we closely follow the presentation in [334, 335]; then discuss more advanced preintegration techniques in Section 11.2.3. As usual, we postpone the discussion of recent works on the topic to Section 11.5.

### 11.2.1 Motion Integration

In this section, we start by inferring the motion of the robot from IMU measurements. For this purpose we introduce the following kinematic model [787, 318]:

$$\dot{\mathbf{R}}_b^w = \mathbf{R}_b^w (\boldsymbol{\omega}_b^b)^\wedge, \quad \dot{\mathbf{v}}^w = \mathbf{a}^w, \quad \dot{\mathbf{p}}^w = \mathbf{v}^w, \quad (11.8)$$

which describes the evolution of the rotation  $\mathbf{R}_b^w$ , translation  $\mathbf{p}^w$ , and velocity  $\mathbf{v}^w$  of the body frame  $\mathcal{F}^b$  with respect to the world frame  $\mathcal{F}^w$ .

The state at time  $t + \Delta t$ , where  $\Delta t$  is the IMU sampling period, is obtained by

integrating (11.8):

$$\mathbf{R}_b^w(t + \Delta t) = \mathbf{R}_b^w(t) \text{Exp} \left( \int_t^{t+\Delta t} \boldsymbol{\omega}_b^b(\tau) d\tau \right) \quad (11.9)$$

$$\begin{aligned} \mathbf{v}^w(t + \Delta t) &= \mathbf{v}^w(t) + \int_t^{t+\Delta t} \mathbf{a}^w(\tau) d\tau \\ \mathbf{p}^w(t + \Delta t) &= \mathbf{p}^w(t) + \int_t^{t+\Delta t} \mathbf{v}^w(\tau) d\tau \end{aligned} \quad (11.10)$$

where in the first equation we assumed that the direction of the angular velocity  $\boldsymbol{\omega}_b^b$  does not change in the interval  $[t, t + \Delta t]$ .<sup>4</sup> Further assuming that  $\mathbf{a}^w$  and  $\boldsymbol{\omega}_b^b$  remain constant in the time interval  $[t, t + \Delta t]$ , we can write:

$$\begin{aligned} \mathbf{R}_b^w(t + \Delta t) &= \mathbf{R}_b^w(t) \text{Exp} (\boldsymbol{\omega}_b^b(t) \Delta t) \\ \mathbf{v}^w(t + \Delta t) &= \mathbf{v}^w(t) + \mathbf{a}^w(t) \Delta t \\ \mathbf{p}^w(t + \Delta t) &= \mathbf{p}^w(t) + \mathbf{v}^w(t) \Delta t + \frac{1}{2} \mathbf{a}^w(t) \Delta t^2. \end{aligned} \quad (11.11)$$

More generally, eq. (11.11) can be understood as applying Euler integration numerically solve the integrals in (11.9). Using Eqs. (11.1)-(11.2), we can write  $\mathbf{a}^w$  and  $\boldsymbol{\omega}_b^b$  as functions of the IMU measurements, hence (11.11) becomes

$$\begin{aligned} \mathbf{R}(t + \Delta t) &= \mathbf{R}(t) \text{Exp} ((\tilde{\boldsymbol{\omega}}(t) - \mathbf{b}^g(t) - \boldsymbol{\eta}^{gd}(t)) \Delta t) \\ \mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \mathbf{g} \Delta t + \mathbf{R}(t) (\tilde{\mathbf{a}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t)) \Delta t \\ \mathbf{p}(t + \Delta t) &= \mathbf{p}(t) + \mathbf{v}(t) \Delta t + \frac{1}{2} \mathbf{g} \Delta t^2 + \frac{1}{2} \mathbf{R}(t) (\tilde{\mathbf{a}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t)) \Delta t^2, \end{aligned} \quad (11.12)$$

where we dropped the coordinate frame subscripts for readability (the notation should be unambiguous from now on). This numeric integration of the velocity and position assumes a constant orientation  $\mathbf{R}(t)$  for the time of integration between two measurements, which is not an exact solution of the differential equation (11.8) for measurements with non-zero rotation rate. In practice, the use of a high-rate IMU mitigates the effects of this approximation. We adopt the integration scheme (11.12) as it is simple and amenable for modeling and uncertainty propagation, and then discuss more advanced integration techniques in Section 11.2.3. The covariance of the discrete-time noise  $\boldsymbol{\eta}^{gd}$  is a function of the sampling rate and relates to the continuous-time noise  $\boldsymbol{\eta}^g$  via  $\text{Cov}(\boldsymbol{\eta}^{gd}(t)) = \frac{1}{\Delta t} \text{Cov}(\boldsymbol{\eta}^g(t))$ . The same relation holds for  $\boldsymbol{\eta}^{ad}$  (*cf.*, [232, Appendix]).

While Eq. (11.12) could be readily seen as a probabilistic constraint in a factor graph, it would require including states in the factor graph at high rate. Intuitively, Eq. (11.12) relates states at time  $t$  and  $t + \Delta t$ , where  $\Delta t$  is the sampling period of the IMU, hence we would have to add new states in the estimation at every new IMU measurement [503].

<sup>4</sup> We provide a more general expression for the rotation integration in eq. (11.35) below.

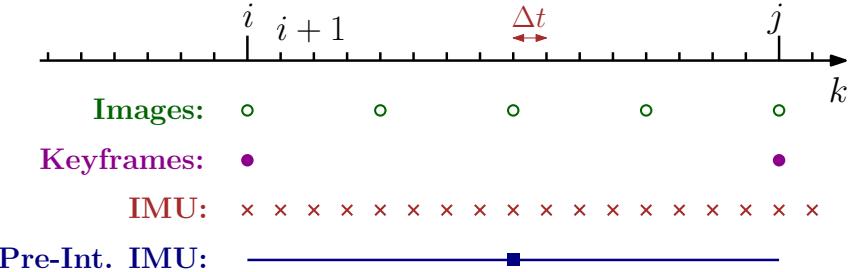


Figure 11.1 Different rates for IMU and camera. From [335] (©2016 IEEE).

We can try to avoid this issue by integrating over longer time intervals. In particular, if we assume that we already have a factor graph modeling other sensor measurements in our problem (*e.g.*, the vision measurements in Chapter 7), we can use the expression (11.12) and integrate IMU measurements between two temporally consecutive states in our factor graph. We are going to refer to these states as “keyframe states”.<sup>5</sup> Iterating the IMU integration (11.12) for all  $\Delta t$  intervals between two consecutive keyframes at times  $t_i$  and  $t_j$  (*c.f.*, Fig. 11.1), we get:<sup>6</sup>

$$\begin{aligned} \mathbf{R}_j &= \mathbf{R}_i \prod_{k=i}^{j-1} \text{Exp} \left( \left( \tilde{\omega}_k - \mathbf{b}_k^g - \boldsymbol{\eta}_k^{gd} \right) \Delta t \right), \\ \mathbf{v}_j &= \mathbf{v}_i + \mathbf{g} \Delta t_{ij} + \sum_{k=i}^{j-1} \mathbf{R}_k \left( \tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t \\ \mathbf{p}_j &= \mathbf{p}_i + \sum_{k=i}^{j-1} \left[ \mathbf{v}_k \Delta t + \frac{1}{2} \mathbf{g} \Delta t^2 + \frac{1}{2} \mathbf{R}_k \left( \tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t^2 \right] \end{aligned} \quad (11.13)$$

where we introduced the shorthands  $\Delta t_{ij} \doteq \sum_{k=i}^{j-1} \Delta t$  and  $(\cdot)_i \doteq (\cdot)(t_i)$  for readability. While Eq. (11.13) already provides an estimate of the motion between time  $t_i$  and  $t_j$ , it has the drawback that the integration in (11.13) has to be repeated whenever the linearization point at time  $t_i$  changes [643] (*e.g.*, at each iteration of a Gauss-Newton solver). For instance, a change in the rotation  $\mathbf{R}_i$  implies a change in all future rotations  $\mathbf{R}_k$ ,  $k = i, \dots, j-1$ , and makes necessary to re-evaluate summations and products in (11.13).

<sup>5</sup> We use this terminology, since in many applications involving IMUs and cameras, the states in the factor graph are added at a subset of the camera frames, namely the *keyframes* (*cf.* Chapter 7). However, this term is used without loss of generality here, and one can decide to instantiate keyframe states arbitrarily (*e.g.*, at every camera frame, LiDAR scans, every “n” IMU measurements, etc.).

<sup>6</sup> For simplicity, we assume that the IMU is synchronized with the other sensors, and IMU measurements are sampled at time  $t_i$  and  $t_j$ . In practice, one can interpolate measurements to approximate the case where IMU measurements are exactly sampled at time  $t_i$  and  $t_j$ ; see Section 11.4.3 for further discussion about temporal synchronization.

### 11.2.2 IMU Preintegration on Manifold

Here we show that a small manipulation of the motion integration results (11.13) allows computing relative measurements between states at time  $t_i$  and  $t_j$  that do not need to be recomputed when the linearization point changes. The key insight is to express measurements in a local frame (such that they do not change when the global state estimate of the robot changes) and isolating the contribution of gravity (which again carries information about the global frame). This process leads to computing the so called *preintegrated IMU measurements*, which constrain the motion between consecutive states in the factor graph.

Towards this goal, we rearrange the terms in (11.13) and define the following relative motion increments that are independent of the pose and velocity at  $t_i$ :

$$\begin{aligned} \Delta \mathbf{R}_{ij} &\doteq \mathbf{R}_i^\top \mathbf{R}_j = \prod_{k=i}^{j-1} \text{Exp} \left( \left( \tilde{\boldsymbol{\omega}}_k - \mathbf{b}_k^g - \boldsymbol{\eta}_k^{gd} \right) \Delta t \right) \\ \Delta \mathbf{v}_{ij} &\doteq \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g} \Delta t_{ij}) = \sum_{k=i}^{j-1} \Delta \mathbf{R}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad}) \Delta t \\ \Delta \mathbf{p}_{ij} &\doteq \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2) \\ &= \sum_{k=i}^{j-1} \left[ \Delta \mathbf{v}_{ik} \Delta t + \frac{1}{2} \Delta \mathbf{R}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad}) \Delta t^2 \right], \end{aligned} \quad (11.14)$$

where  $\Delta \mathbf{R}_{ik} \doteq \mathbf{R}_i^\top \mathbf{R}_k$  and  $\Delta \mathbf{v}_{ik} \doteq \mathbf{R}_i^\top (\mathbf{v}_k - \mathbf{v}_i - \mathbf{g} \Delta t_{ik})$ . We highlight that, in contrast to the “delta” rotation  $\Delta \mathbf{R}_{ij}$ , neither  $\Delta \mathbf{v}_{ij}$  nor  $\Delta \mathbf{p}_{ij}$  correspond to the true *physical* change in velocity and position but are defined in a way that make the right-hand side of (11.14) independent from the state at time  $i$  as well as gravitational effects. Indeed, we will be able to compute the right-hand side of (11.14) directly from the inertial measurements between the two keyframes.

Unfortunately, summations and products in (11.14) are still function of the bias estimate. We tackle this problem in two steps. In Section 11.2.2.1, we assume  $\mathbf{b}_i$  is known; then, in Section 11.2.2.3 we show how to avoid repeating the integration when the bias estimate changes. In both cases, we assume the biases remain constant between times  $t_i$  and  $t_j$ :

$$\mathbf{b}_i^g = \mathbf{b}_{i+1}^g = \dots = \mathbf{b}_{j-1}^g, \quad \mathbf{b}_i^a = \mathbf{b}_{i+1}^a = \dots = \mathbf{b}_{j-1}^a. \quad (11.15)$$

#### 11.2.2.1 Preintegrated IMU Measurements

Equation (11.14) relates the states of keyframes  $i$  and  $j$  (left-hand side) to the measurements (right-hand side). In this sense, it can be already understood as a measurement model. Unfortunately, it has a fairly intricate dependence on the measurement noise and this complicates a direct application of MAP estimation; intuitively, the MAP estimator requires to clearly define the densities (and their log-likelihood) of the measurements. In this section we manipulate (11.14) so to make

*preintegrated IMU measurements*

easier the derivation of the measurement log-likelihood. More concretely, we isolate the noise terms of the individual inertial measurements in (11.14). As discussed above, within this section assume that the bias at time  $t_i$  is known.

Let us start with the rotation increment  $\Delta\mathbf{R}_{ij}$  in (11.14). Towards this goal, we use the following properties of the exponential map for SO(3) (*cf.* Chapter 2):

$$\text{Exp}(\phi + \delta\phi) \approx \text{Exp}(\phi) \text{Exp}(\mathbf{J}_r(\phi)\delta\phi), \quad (11.16)$$

$$\text{Exp}(\phi) \mathbf{R} = \mathbf{R} \text{Exp}(\mathbf{R}^\top\phi). \quad (11.17)$$

where the first relations is a first-order approximation of the exponential of a sum, and the second can be derived from the group's adjoint representation.

Using (11.16) and (11.17), we rearrange the terms in the expression of  $\Delta\mathbf{R}_{ij}$  in (11.14), by “moving” the noise to the end:

$$\begin{aligned} \Delta\mathbf{R}_{ij} &\stackrel{\text{eq.(11.16)}}{\simeq} \prod_{k=i}^{j-1} \left[ \text{Exp}((\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g)\Delta t) \text{Exp}\left(-\mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t\right) \right] \\ &\stackrel{\text{eq.(11.17)}}{=} \Delta\tilde{\mathbf{R}}_{ij} \prod_{k=i}^{j-1} \text{Exp}\left(-\Delta\tilde{\mathbf{R}}_{k+1,j}^\top \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t\right) \\ &\doteq \Delta\tilde{\mathbf{R}}_{ij} \text{Exp}(-\delta\phi_{ij}) \end{aligned} \quad (11.18)$$

*Preintegrated measurement* with  $\mathbf{J}_r^k \doteq \mathbf{J}_r^k((\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g)\Delta t)$ . In the last line of (11.18), we defined the *preintegrated rotation measurement*  $\Delta\tilde{\mathbf{R}}_{ij} \doteq \prod_{k=i}^{j-1} \text{Exp}((\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g)\Delta t)$ , and its noise  $\delta\phi_{ij}$ , which will be further analysed in the next section.

Similarly, we can manipulate the velocity and position equations in (11.14) by using the following relations:

$$\exp(\phi^\wedge) \approx \mathbf{I} + \phi^\wedge, \quad (11.19)$$

$$\mathbf{a}^\wedge \mathbf{b} = -\mathbf{b}^\wedge \mathbf{a}, \quad \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^3, \quad (11.20)$$

where the first relation is a first-order approximation of the exponential map at the origin, while the second is a property of the wedge operator of a vector.

Substituting (11.18) back into the expression of  $\Delta\mathbf{v}_{ij}$  in (11.14), using the first-order approximation (11.19) for  $\text{Exp}(-\delta\phi_{ij})$ , and dropping higher-order noise terms, we obtain:

$$\begin{aligned} \Delta\mathbf{v}_{ij} &\stackrel{\text{eq.(11.19)}}{\simeq} \sum_{k=i}^{j-1} \Delta\tilde{\mathbf{R}}_{ik} (\mathbf{I} - \delta\phi_{ik}^\wedge) (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a) \Delta t - \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t \\ &\stackrel{\text{eq.(11.20)}}{=} \Delta\tilde{\mathbf{v}}_{ij} + \sum_{k=i}^{j-1} \left[ \Delta\tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a)^\wedge \delta\phi_{ik} \Delta t - \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t \right] \\ &\doteq \Delta\tilde{\mathbf{v}}_{ij} - \delta\mathbf{v}_{ij} \end{aligned} \quad (11.21)$$

*Preintegrated velocity* where we defined the *preintegrated velocity measurement*  $\Delta\tilde{\mathbf{v}}_{ij} \doteq \sum_{k=i}^{j-1} \Delta\tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a) \Delta t$

and its noise  $\delta\mathbf{v}_{ij}$ .

Similarly, substituting (11.18) and (11.21) in the expression of  $\Delta\mathbf{p}_{ij}$  in (11.14), and using the first-order approximation (11.19), we obtain:

$$\begin{aligned} \Delta\mathbf{p}_{ij} &\stackrel{\text{eq.(11.19)}}{\simeq} \sum_{k=i}^{j-1} \left[ (\Delta\tilde{\mathbf{v}}_{ik} - \delta\mathbf{v}_{ik}) \Delta t + \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} (\mathbf{I} - \delta\phi_{ik}^\wedge) (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a) \Delta t^2 \right. \\ &\quad \left. - \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t^2 \right] \\ &\stackrel{\text{eq.(11.20)}}{=} \Delta\tilde{\mathbf{p}}_{ij} + \sum_{k=i}^{j-1} \left[ -\delta\mathbf{v}_{ik} \Delta t + \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a)^\wedge \delta\phi_{ik} \Delta t^2 \right. \\ &\quad \left. - \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t^2 \right] \\ &\doteq \Delta\tilde{\mathbf{p}}_{ij} - \delta\mathbf{p}_{ij}, \end{aligned} \quad (11.22)$$

where we defined the *preintegrated position measurement*  $\Delta\tilde{\mathbf{p}}_{ij}$  and its noise  $\delta\mathbf{p}_{ij}$ . Preintegrated measurement

Substituting the expressions (11.18), (11.21), (11.22) back in the original definition of  $\Delta\mathbf{R}_{ij}, \Delta\mathbf{v}_{ij}, \Delta\mathbf{p}_{ij}$  in (11.14), we finally get our *preintegrated measurement model* (remember  $\text{Exp}(-\delta\phi_{ij})^\top = \text{Exp}(\delta\phi_{ij})$ ):

$$\begin{aligned} \Delta\tilde{\mathbf{R}}_{ij} &= \mathbf{R}_i^\top \mathbf{R}_j \text{Exp}(\delta\phi_{ij}) \\ \Delta\tilde{\mathbf{v}}_{ij} &= \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) + \delta\mathbf{v}_{ij} \\ \Delta\tilde{\mathbf{p}}_{ij} &= \mathbf{R}_i^\top \left( \mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2 \right) + \delta\mathbf{p}_{ij} \end{aligned} \quad (11.23)$$

where our compound measurements are written as a function of the (to-be-estimated) state “plus” a random noise, described by the random vector  $[\delta\phi_{ij}^\top, \delta\mathbf{v}_{ij}^\top, \delta\mathbf{p}_{ij}^\top]^\top$ . position

In summary, in this section we manipulated the measurement model (11.14) and rewrote it as (11.23). The advantage of the measurements in eq. (11.23) is that, for a suitable distribution of the noise, they can be used directly to instantiate factors between states at time  $t_i$  and  $t_j$  in our factor graph. The nature of the noise terms is discussed in the following section.

### 11.2.2.2 Noise Propagation

In this section we derive the statistics of the noise vector  $[\delta\phi_{ij}^\top, \delta\mathbf{v}_{ij}^\top, \delta\mathbf{p}_{ij}^\top]^\top$ . While we already observed that it is convenient to approximate the noise vector to be zero-mean Normally distributed, it is of paramount importance to accurately model the noise covariance. In this section, we therefore provide a derivation of the covariance  $\boldsymbol{\Sigma}_{ij}$  of the preintegrated measurements:

$$\boldsymbol{\eta}_{ij}^\Delta \doteq [\delta\phi_{ij}^\top, \delta\mathbf{v}_{ij}^\top, \delta\mathbf{p}_{ij}^\top]^\top \sim \mathcal{N}(\mathbf{0}_{9 \times 1}, \boldsymbol{\Sigma}_{ij}). \quad (11.24)$$

We first consider the preintegrated rotation noise  $\delta\phi_{ij}$ . Recall from (11.18) that

$$\text{Exp}(-\delta\phi_{ij}) \doteq \prod_{k=i}^{j-1} \text{Exp} \left( -\Delta\tilde{\mathbf{R}}_{k+1j}^\top \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t \right). \quad (11.25)$$

Taking the Log on both sides and changing signs, we get:

$$\delta\phi_{ij} = -\text{Log}\left(\prod_{k=i}^{j-1} \text{Exp}\left(-\Delta\tilde{\mathbf{R}}_{k+1,j}^T \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t\right)\right). \quad (11.26)$$

Next, we use the following first-order approximation holds for SO(3) logarithm:

$$\text{Log}(\text{Exp}(\phi) \text{Exp}(\delta\phi)) \approx \phi + \mathbf{J}_r^{-1}(\phi)\delta\phi. \quad (11.27)$$

where  $\mathbf{J}_r^{-1}(\phi)$  is the inverse of the right Jacobian. Repeated application of (11.27) (recall that  $\boldsymbol{\eta}_k^{gd}$  as well as  $\delta\phi_{ij}$  are small rotation noises, hence the right Jacobians are close to the identity) produces:

$$\delta\phi_{ij} \simeq \sum_{k=i}^{j-1} \Delta\tilde{\mathbf{R}}_{k+1,j}^T \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t \quad (11.28)$$

Up to first order, the noise  $\delta\phi_{ij}$  is zero-mean and Gaussian, as it is a linear combination of zero-mean noise terms  $\boldsymbol{\eta}_k^{gd}$ .

Dealing with the noise terms  $\delta\mathbf{v}_{ij}$  and  $\delta\mathbf{p}_{ij}$  is now easy: these are linear combinations of the acceleration noise  $\boldsymbol{\eta}_k^{ad}$  and the preintegrated rotation noise  $\delta\phi_{ij}$ , hence they are also zero-mean and Gaussian. Simple manipulation leads to:

$$\begin{aligned} \delta\mathbf{v}_{ij} &\simeq \sum_{k=i}^{j-1} \left[ -\Delta\tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a)^\wedge \delta\phi_{ik} \Delta t + \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t \right] \\ \delta\mathbf{p}_{ij} &\simeq \sum_{k=i}^{j-1} \left[ \delta\mathbf{v}_{ik} \Delta t - \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{a}}_k - \mathbf{b}_i^a)^\wedge \delta\phi_{ik} \Delta t^2 + \frac{1}{2} \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t^2 \right] \end{aligned} \quad (11.29)$$

where the relations are valid up to the first order.

Eqs. (11.28)-(11.29) express the preintegrated noise  $\boldsymbol{\eta}_{ij}^\Delta$  as a linear function of the IMU measurement noise  $\boldsymbol{\eta}_k^d \doteq [\boldsymbol{\eta}_k^{gd}, \boldsymbol{\eta}_k^{ad}]$ ,  $k = 1, \dots, j-1$ . Therefore, from the knowledge of the covariance of  $\boldsymbol{\eta}_k^d$  (given in the IMU specifications), we can compute the covariance of  $\boldsymbol{\eta}_{ij}^\Delta$ , namely  $\boldsymbol{\Sigma}_{ij}$ , by a simple linear propagation.

An extended derivation of the noise propagation can be found in [335], which also provides an iterative expression to compute the covariance by incrementally adding new measurements as they are collected. The iterative computation leads to simpler expressions and is more amenable for online inference.

### 11.2.2.3 Incorporating Bias Updates

In the previous section, we assumed that the bias, say  $\{\bar{\mathbf{b}}_i^a, \bar{\mathbf{b}}_i^g\}$ , that is used during preintegration between  $k = i$  and  $k = j$  is correct and does not change. However, more likely, the bias estimate changes by a small amount  $\delta\mathbf{b}$  during optimization. One solution would be to recompute the delta measurements when the bias changes; however, that is computationally expensive. Instead, given a bias update  $\mathbf{b} \leftarrow \bar{\mathbf{b}} + \delta\mathbf{b}$ , we can update the delta measurements using a first-order expansion:

$$\begin{aligned}\Delta \tilde{\mathbf{R}}_{ij}(\mathbf{b}_i^g) &\simeq \Delta \tilde{\mathbf{R}}_{ij}(\bar{\mathbf{b}}_i^g) \operatorname{Exp}\left(\frac{\partial \Delta \tilde{\mathbf{R}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g\right) \\ \Delta \tilde{\mathbf{v}}_{ij}(\mathbf{b}_i^g, \mathbf{b}_i^a) &\simeq \Delta \tilde{\mathbf{v}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{\mathbf{v}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}_i^g + \frac{\partial \Delta \tilde{\mathbf{v}}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}_i^a \\ \Delta \tilde{\mathbf{p}}_{ij}(\mathbf{b}_i^g, \mathbf{b}_i^a) &\simeq \Delta \tilde{\mathbf{p}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{\mathbf{p}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}_i^g + \frac{\partial \Delta \tilde{\mathbf{p}}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}_i^a\end{aligned}\quad (11.30)$$

This is similar to the bias correction in [706] but operates directly on SO(3). The Jacobians  $\{\frac{\partial \Delta \tilde{\mathbf{R}}_{ij}}{\partial \mathbf{b}^g}, \frac{\partial \Delta \tilde{\mathbf{v}}_{ij}}{\partial \mathbf{b}^g}, \dots\}$  (computed at  $\bar{\mathbf{b}}_i$ , the bias estimate at integration time) describe how the measurements change due to a change in the bias estimate. The Jacobians remain constant and can be precomputed during the preintegration. The derivation of the Jacobians is very similar to the one we used in Section 11.2.2.1 to express the measurements as a large value *plus* a small perturbation and is given in [335].

#### 11.2.2.4 Preintegrated IMU Factors and Bias Models

Given the preintegrated measurement model in (11.23) and since measurement noise is zero-mean and Gaussian (with covariance  $\Sigma_{ij}$ ) up to first order (11.24), it is now easy to write the residual errors  $\mathbf{r}_{\mathcal{I}_{ij}} \doteq [\mathbf{r}_{\Delta \mathbf{R}_{ij}}^\top, \mathbf{r}_{\Delta \mathbf{v}_{ij}}^\top, \mathbf{r}_{\Delta \mathbf{p}_{ij}}^\top]^\top \in \mathbb{R}^9$ , which will appear in the factor graph optimization:

$$\begin{aligned}\mathbf{r}_{\Delta \mathbf{R}_{ij}} &\doteq \operatorname{Log}\left(\left(\Delta \tilde{\mathbf{R}}_{ij}(\bar{\mathbf{b}}_i^g) \operatorname{Exp}\left(\frac{\partial \Delta \tilde{\mathbf{R}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g\right)\right)^\top \mathbf{R}_i^\top \mathbf{R}_j\right) \\ \mathbf{r}_{\Delta \mathbf{v}_{ij}} &\doteq \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g} \Delta t_{ij}) \\ &\quad - \left[\Delta \tilde{\mathbf{v}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{\mathbf{v}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g + \frac{\partial \Delta \tilde{\mathbf{v}}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}^a\right] \\ \mathbf{r}_{\Delta \mathbf{p}_{ij}} &\doteq \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2) \\ &\quad - \left[\Delta \tilde{\mathbf{p}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{\mathbf{p}}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g + \frac{\partial \Delta \tilde{\mathbf{p}}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}^a\right],\end{aligned}\quad (11.31)$$

in which we also included the bias updates of Eq. (11.30). These terms can be *IMU factors* readily added to the factor graph by adding the term  $\|\mathbf{r}_{\mathcal{I}_{ij}}\|_{\Sigma_{ij}}^2$  to the objective of the minimization.

When presenting the IMU model (11.1)-(11.2), we said that biases are slowly time-varying quantities. Hence, we model them with a “Brownian motion”, *i.e.*,

*Brownian motion*

$$\dot{\mathbf{b}}^g(t) = \boldsymbol{\eta}^{bg}, \quad \dot{\mathbf{b}}^a(t) = \boldsymbol{\eta}^{ba}. \quad (11.32)$$

Integrating (11.32) over the time interval  $[t_i, t_j]$  between two consecutive keyframes  $i$  and  $j$  we get:

$$\mathbf{b}_j^g = \mathbf{b}_i^g + \boldsymbol{\eta}^{bgd}, \quad \mathbf{b}_j^a = \mathbf{b}_i^a + \boldsymbol{\eta}^{bad}, \quad (11.33)$$

where, as done before, we use the shorthand  $\mathbf{b}_i^g \doteq \mathbf{b}^g(t_i)$ , and we define the discrete

noises  $\boldsymbol{\eta}^{bgd}$  and  $\boldsymbol{\eta}^{bad}$ , which have zero mean and covariance  $\boldsymbol{\Sigma}^{bgd} \doteq \Delta t_{ij} \text{Cov}(\boldsymbol{\eta}^{bg})$  and  $\boldsymbol{\Sigma}^{bad} \doteq \Delta t_{ij} \text{Cov}(\boldsymbol{\eta}^{ba})$ , respectively (cf. [232, Appendix]).

The model (11.33) can be readily included in our factor graph, as a further additive term in the objective function, for all consecutive keyframes:

$$\|\mathbf{r}_{\mathbf{b}_{ij}}\|^2 \doteq \|\mathbf{b}_j^g - \mathbf{b}_i^g\|_{\boldsymbol{\Sigma}^{bgd}}^2 + \|\mathbf{b}_j^a - \mathbf{b}_i^a\|_{\boldsymbol{\Sigma}^{bad}}^2 \quad (11.34)$$

### 11.2.3 Advanced Preintegration Techniques

In this section, we look at some limitations of the standard preintegration approach and explore newer alternatives. We first look at the underlying signal and motion assumptions embedded in (11.14). Then we go through various works that alleviate these assumptions leading to more accurate preintegrated measurements, thus improved localization and mapping accuracy when used for aided inertial navigation. Note that we do not detail the derivation of each of the methods and invite the reader to refer to the corresponding papers for a more extensive treatment.

#### 11.2.3.1 Numerical Integration Accuracy

As described in the previous section, standard preintegration relies on the Euler method to integrate inertial signals into rotation, velocity, and position pseudo-measurements at discrete times. This approach is fast and efficient but introduces integration error (hence drift) in the preintegration process. In short, the Euler method consists in applying the rectangle rule to a signal to numerically obtain its integral. As illustrated in Figure 11.2 (left), it means that the signal is approximated with piecewise constant chunks sampled at a given frequency. In the context of inertial systems, the samples correspond to accelerometer or gyroscope measurements.

*sampling frequency*

With a fairly low sampling frequency, the piecewise constant assumption does not accurately represent the input signal. Consequently, the double integration rapidly accumulates error (Figure 11.2 (top right)). A possible workaround consists in increasing the sampling frequency of the signal (Figure 11.2 (bottom)). However, in real-world inertial navigation problems, the sampling frequency is limited by the hardware characteristics of the inertial sensor.

*GP regression*

In [629], the authors propose to use GP regression<sup>7</sup> as a mean to virtually upsample the input signal at any chosen timestamp for both the gyroscope and accelerometer data. While improving over standard preintegration, such an approach still performs numerical integration based on the piecewise constant assumption and does not fully leverage the continuous nature of GP models. Below, we review more sophisticated integration approaches.

<sup>7</sup> GP regression is a non-parametric, probabilistic approach for interpolation. We invite the reader to refer to [906] for a deeper understanding of GP regression.

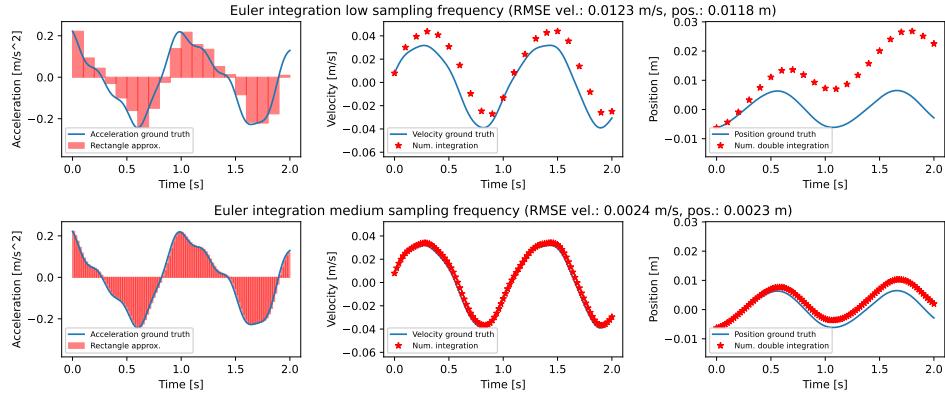


Figure 11.2 Example of Euler integration (with known initial conditions) using low (top row) and high (bottom row) sampling frequencies.

### 11.2.3.2 Continuous Acceleration Preintegration

Another way to reduced the integration error is to leverage continuous-time representations—which are not limited to discrete timestamps—that better approximate the true inertial signals and perform analytical integration. Atop the gain in accuracy, continuous-time representations allow for asynchronous query of the preintegrated measurements. This is especially useful when performing inertial-aided state estimation with other sensors that are not hardware-synchronized or that have completely asynchronous sampling processes (*e.g.*, event cameras).

A challenging component of preintegration is dealing with the space of rotations. The non-commutative nature of rotation operations prevents the use of numerous tools available for classic Riemann integration. Accordingly, several works have dissociated the rotational and translation parts of preintegration. In this subsection, we first explore the translation component of preintegration using continuous-time representations while assuming solved rotation integration. Continuous-time integration over the rotation space will be addressed in the following subsection.

In [300], after using a zeroth-order integrator [1102] to integrate the gyroscope measurements, the authors present a continuous formulation of the velocity and position preintegrated measurements by solving the continuous-time system of differential equation (LTV) assuming constant accelerometer measurements or constant *local* acceleration (the two different models are presented in [300]). Compared with the standard preintegration [335] that considers constant global acceleration, the work [300] demonstrates that the assumption of constant local acceleration is more representative of real scenarios leading to an overall VIO accuracy improvement of around 5% on the EuRoc dataset [131] over both the standard preintegration and the constant accelerometer measurement model.

In order to loosen the constant acceleration motion model assumptions, one can

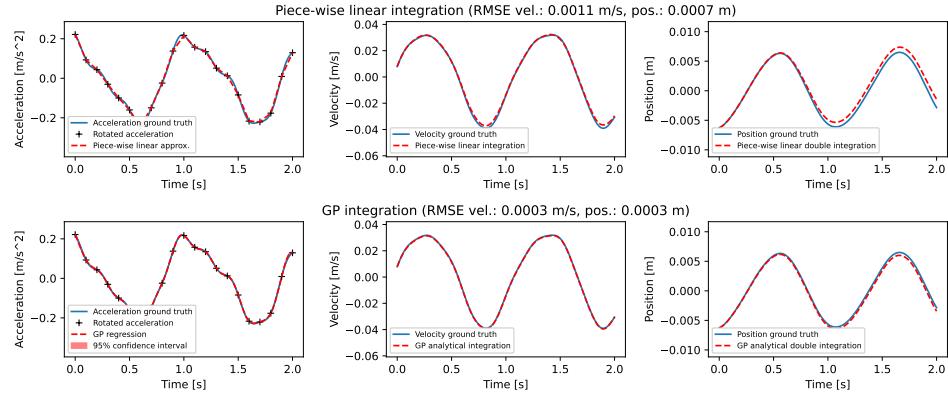


Figure 11.3 Top row: Continuous integration with piecewise-linear approximation (corresponding to constant-jerk motion assumption). Bottom row: Model-free integration with Gaussian Process regression.

approximate the input data with analytically integrable functions. Assuming that the rotational part of the preintegration is solved, the authors in [630] represent the rotation-corrected accelerometer measurements  $\hat{\mathbf{a}}_k$ , defined as  $\hat{\mathbf{a}}_k = \Delta \mathbf{R}_{ik} \tilde{\mathbf{a}}_k$ , in a continuous manner. We show in Figure 11.3 the accuracy gain of integration with both piecewise-linear and GP-based continuous representations compared to the Euler method shown in Figure 11.2. With the piecewise-linear approximation, the first integral (from  $\hat{\mathbf{a}}_k$  to  $\Delta \mathbf{v}_{ik}$ ) corresponds to the classic trapezoidal rule for numerical integration. This model can be interpreted as a constant-jerk motion model and already provides a significant accuracy gain compared to the Euler method. Going further in the quest for model-less integration, using GP regression with  $\hat{\mathbf{a}} \sim \mathcal{GP}(0, k_{\mathbf{a}}(t, t')\mathbf{I})$  and  $k_{\mathbf{a}}(t, t')$  is the square exponential covariance kernel function, the direct analytical inference of the integral (and double integral) of  $\hat{\mathbf{a}}$  is enabled by the application of linear operators on GPs [966]. Accordingly, the method does not rely on any explicit motion model as the square exponential kernel is infinitely differentiable. The bottom row of Figure 11.3 shows how the non-parametric GP model improves the integration accuracy compared to the piecewise-linear method. Note that the kernel's hyperparameters control the smoothness of the signal and can be learned from the data or be set with an educated guess.

#### 11.2.3.3 Continuous Rotation Preintegration

Looking at the accuracy gain brought by continuous representations for the translation and velocity preintegration, we naturally want to extend the concept to the rotation part. However, integrating over the space of rotations is challenging as the rotations  $\mathbf{R}$  belong to the  $SO(3)$  Lie group, which is not an Euclidean space. Properties like the commutativity of the group operation do not hold for rotations.

Indeed, the product integral

$$\mathbf{R}_b^w(t + \Delta t) = \mathbf{R}_b^w(t) \prod_t^{t+\Delta t} \text{Exp}(\boldsymbol{\omega}_b^b(\tau))^{d\tau} \quad (11.35)$$

that solves the kinematic model (11.8) does not have a known generic solution [109] and novel approaches are required to perform continuous model-less integration over the space of rotations.

In response to this challenge, the authors of [628] propose to leverage the rotation vector representation  $\mathbf{r}(t)$  in the Lie algebra (with  $\mathbf{R}(t) = \text{Exp}(\mathbf{r}(t))$ ) as a linear vector space to perform continuous integration using linear tools. In that space, the system's dynamics is

$$\dot{\mathbf{r}} = (\mathbf{J}_r(\mathbf{r}))^{-1} \boldsymbol{\omega}_b^b, \quad (11.36)$$

where  $\mathbf{J}_r(\mathbf{r})$  is the right-hand Jacobian of SO(3) evaluated at  $\mathbf{r}$ . Unfortunately, neither  $\mathbf{r}$  nor  $\dot{\mathbf{r}}$  are directly observed by the IMU. The key idea of [628] is to model  $\dot{\mathbf{r}}$  with a GP and a set of virtual observations  $\dot{\mathbf{r}}_{t_i}$  to represent the continuous rotation vector function  $\mathbf{r}$  via the use of linear operators on GPs. Intuitively, the virtual observations can be interpreted as control points of the continuous rotational dynamics. These are estimated through a non-linear least-square optimisation problem with residuals based on (11.36) and the gyroscope measurements as observations of  $\boldsymbol{\omega}_b^b$ . This results in a model-less approach to continuous rotation preintegration and yields accuracy improvements of at least one order of magnitude over the standard discrete preintegration.

This continuous approach shares a lot of similarities with the STEAM continuous-time state estimation detailed in [47] and mentioned in Chapter 2 as both operate in the Lie algebra to perform GP-based interpolation. A major difference is the use of the square exponential kernel that results in a dense linear system, compared to the sparse Markovian approach used in STEAM. However, for the sake of IMU preintegration the length of an integration window is generally short enough that solving a dense system is not an issue. The concept of optimized inducing values is extended in [376] to also estimate the rotation-corrected acceleration along with the rotation vector. This allows to correlate the rotation and translation parts of the preintegrated measurement covariance matrix.

### 11.3 Observability of Aided Inertial Navigation

As we mentioned earlier in this chapter, due to measurement noise, biases, and inaccuracies of numerical integration, pure inertial odometry may drift quickly, in particular when using low-fidelity inertial sensors. A common approach to reduce the drift is to pair the IMU with exteroceptive sensors, *e.g.*, cameras or LiDARs, leading to *aided INS (AINS)*. In many cases, the introduction of exteroceptive sensors further increases the size of the state we have to estimate, *e.g.*, by adding

extra variables corresponding to external landmarks, hence a natural question to ask is whether the sensor data is *sufficient* to unambiguously estimate the SLAM state of the system. This is the goal of the *observability analysis*, which ascertains whether the information provided by the available measurements is sufficient for estimating the state/parameters without ambiguity [117, 453].

The observability analysis is typically performed by deriving linearized measurement models and computing the *observability matrix*, which is closely related to the Fisher information (and covariance) matrix of the state estimate [487, 485] (*cf.* Chapter 6). When a system is observable, the observability matrix is full-rank; if not, as this matrix describes the information available in the measurements, studying its nullspace enables us to gain insights about the directions in the state space along which the estimator lacks sufficient information. The results of the observability analysis can be used to improve estimation consistency [1231, 456, 651], determine the minimal measurements needed to initialize an estimator [456, 734], and also identify degenerate motions that cause additional unobservable directions and should be avoided or alerted if possible in practice [1232]. For these reasons, significant research efforts have been devoted to the observability analysis of AINS [1231], and in particular visual-inertial systems (*e.g.*, [457, 652, 1232]).

In this section we discuss observability properties when the sensors used to aid the IMU produce geometric features, including points, lines, and planes. This general treatment allows discussing observability for a broad range of sensors, including cameras and LiDARs, and understanding degenerate configurations. In particular, Section 11.3.1 introduces linearized models assuming exteroceptive measurements of geometric landmarks, Section 11.3.2 uses these models to perform the observability analysis, and Section 11.3.3 discusses degenerate configurations.

### 11.3.1 Linearized Measurement Models

We describe the measurement models assuming that the sensor (*e.g.*, camera, LiDAR) used to aid the IMU produces geometric features; in other words, we focus on SLAM and odometry front-ends which produce landmark-based representations. While most AINS use point features, in particular when relying on cameras (*e.g.*, [456, 651, 643, 893, 375, 335]), line and plane features can be utilized when available (*e.g.*, [598, 455, 414, 1233]). In such a case, we may need to augment the to-be-estimated state vector with all these different geometric features. Specifically, the AINS state that we are trying to estimate (at each time step) includes both the state of the robot  $\mathbf{x}_b$  and the state of external features  $\mathbf{x}_f^w$  (expressed in the world frame):

$$\mathbf{x} = \{\mathbf{R}_b^w, \mathbf{b}^g, \mathbf{v}^w, \mathbf{b}^a, \mathbf{p}^w, \mathbf{x}_f^w\} \quad (11.37)$$

where  $\mathbf{R}_b^w$  is the rotation of the body frame  $\mathcal{F}^b$  with respect to the world frame  $\mathcal{F}^w$ , and  $\mathbf{p}^w, \mathbf{v}^w$  are the robot position and velocity expressed in the world frame,

respectively, while  $\mathbf{b}^g, \mathbf{b}^a$  are the gyroscope and accelerometer biases in the body frame. The features  $\mathbf{x}_f^w$  can be either points, lines, or planes (or a combination thereof) and are expressed in the world frame.

For the ensuing observability analysis, we need both the system dynamic model (which is related to the accelerations and angular rate measurements of the IMU) and the exteroceptive measurement model. Below, we start by reviewing the INS kinematic model —building on the IMU equations introduced in the previous section— and then consider exteroceptive measurement equations.

#### 11.3.1.1 Linearized IMU Kinematic Model

The IMU-based kinematic model is given by (*cf.* (11.8) and (11.32)):

*linearized  
model*

$$\dot{\mathbf{R}}_b^w = \mathbf{R}_b^w(\boldsymbol{\omega}_b^b)^\wedge, \quad \dot{\mathbf{v}}^w = \mathbf{a}^w, \quad \dot{\mathbf{p}}^w = \mathbf{v}^w, \quad (11.38)$$

$$\dot{\mathbf{b}}^g(t) = \boldsymbol{\eta}^{bg}, \quad \dot{\mathbf{b}}^a(t) = \boldsymbol{\eta}^{ba} \quad (11.39)$$

where  $\boldsymbol{\eta}^{bg}$  and  $\boldsymbol{\eta}^{ba}$  are the zero-mean Gaussian noises driving the gyroscope and accelerometer biases (which are modeled as random walks). In order to perform the observability analysis, we linearize the above nonlinear system and obtain the following continuous-time linearized error-state dynamical system:

$$\dot{\tilde{\mathbf{x}}}(t) \simeq \begin{bmatrix} \mathbf{F}_c(t) & \mathbf{0}_{15 \times n_f} \\ \mathbf{0}_{n_f \times 15} & \mathbf{0}_{n_f} \end{bmatrix} \tilde{\mathbf{x}}(t) + \begin{bmatrix} \mathbf{G}_c(t) \\ \mathbf{0}_{n_f \times 12} \end{bmatrix} \boldsymbol{\eta}(t) =: \mathbf{F}(t)\tilde{\mathbf{x}}(t) + \mathbf{G}(t)\boldsymbol{\eta}(t) \quad (11.40)$$

where the error-state vector  $\tilde{\mathbf{x}} = \{\tilde{\boldsymbol{\theta}}, \tilde{\mathbf{b}}^g, \tilde{\mathbf{v}}^w, \tilde{\mathbf{b}}^a, \tilde{\mathbf{p}}^w, \tilde{\mathbf{x}}_f^w\}$  (expressed as a column vector) represents the deviation from the linearization point (*e.g.*,  $\tilde{\mathbf{b}}^g$  is the change of the bias with respect to the linearization point), and for the rotation component we use the tangent-space representation  $\tilde{\boldsymbol{\theta}}$  at the linearization point.<sup>8</sup> In (11.40),  $n_f$  is the dimension of  $\tilde{\mathbf{x}}_f^w$ ,  $\mathbf{F}_c(t)$  and  $\mathbf{G}_c(t)$  are the continuous-time linearized transition matrix and the noise Jacobian matrix for the IMU state, respectively, and  $\boldsymbol{\eta}(t)$  is the stacked noise, including both  $\boldsymbol{\eta}^{bg}$  and  $\boldsymbol{\eta}^{ba}$  as well as the IMU noise which arises when substituting the actual acceleration and rotation rates in (11.38) with the accelerometer and gyroscope measurements (*cf.* with derivation in Section 11.2.1).

As in practice AINS estimators are typically implemented in discrete time, the discrete-time dynamic model is needed and can be derived by computing its state transition matrix  $\Phi_{(k+1,k)}$  from time  $t_k$  to  $t_{k+1}$ , based on  $\dot{\Phi}_{(k+1,k)} = \mathbf{F}(t_k)\Phi_{(k+1,k)}$

<sup>8</sup> Intuitively, to linearize the rotation variables, we use the fact that we can rewrite any rotation  $\mathbf{R}_b^w$  as a perturbation of the rotation at the linearization point  $\hat{\mathbf{R}}_b^w$ :  $\mathbf{R}_b^w = \hat{\mathbf{R}}_b^w \mathbf{R}_b^w(\tilde{\boldsymbol{\theta}})$ , where  $\tilde{\boldsymbol{\theta}}$  is a suitable tangent-space vector. Then we can use the following small-angle approximation to linearize the expression:  $\mathbf{R}_b^w = \hat{\mathbf{R}}_b^w \mathbf{R}_b^w(\tilde{\boldsymbol{\theta}}) \simeq \hat{\mathbf{R}}_b^w(\mathbf{I} + \tilde{\boldsymbol{\theta}}^\wedge)$ .

with identity as the initial condition:

$$\Phi_{(k+1,k)} = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \Phi_{31} & \Phi_{32} & \mathbf{I}_3 & \Phi_{34} & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \Phi_{51} & \Phi_{52} & \Phi_{53} & \Phi_{54} & \mathbf{I}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_{3 \times n_f} & \mathbf{I}_{n_f} \end{bmatrix} \quad (11.41)$$

where the  $(i,j)$  block  $\Phi_{ij}$  can be found analytically or numerically [456].

### 11.3.1.2 Exteroceptive Measurement Models

Now we present the measurement models of different geometric features and their linearized models, which are essential for the linearized AINS observability analysis.

*range-and-bearing measurements* *mea-* **Point Features.** Consider point feature measurements provided by exteroceptive sensors (such as monocular/stereo camera, acoustic sonar, and LiDAR). These can generally be modeled as range and/or bearing observations, which are functions of the relative position of the feature in the sensor frame  $\mathcal{F}^c$ :

$$\mathbf{z}_p = \underbrace{\begin{bmatrix} \lambda_r & \mathbf{0}_{1 \times 2} \\ \mathbf{0}_{2 \times 1} & \lambda_b \mathbf{I}_2 \end{bmatrix}}_{\Lambda} \begin{bmatrix} z_r \\ \mathbf{z}_b \end{bmatrix} = \Lambda \begin{bmatrix} \|\mathbf{p}_f^c\| + \eta^r \\ h_b(\mathbf{p}_f^c) + \boldsymbol{\eta}^b \end{bmatrix} \quad (11.42)$$

where  $\mathbf{p}_f^c = \mathbf{R}_w^c (\mathbf{p}_f^w - \mathbf{p}_c^w)$  is the position of the feature in the sensor frame, and  $z_r$  and  $\mathbf{z}_b$  denote range and bearing measurements, respectively. In particular,  $h_b(\cdot)$  is a generic bearing measurement function whose actual form depends on the particular sensor used.  $\Lambda$  is a measurement selection matrix, with binary entries  $\lambda_r$  and  $\lambda_b$ ; for example, if  $\lambda_b = 1$  and  $\lambda_r = 1$ , then  $\mathbf{z}_p$  contains both range and bearing measurements.  $\eta^r$  and  $\boldsymbol{\eta}^b$  are the measurement noises and are assumed to be additive for simplicity. Linearizing (11.42) with the chain rule of differentiation at the current state estimate yields the following measurement error equation:

$$\tilde{\mathbf{z}}_p = \mathbf{z}_p - \hat{\mathbf{z}}_p \simeq \Lambda \begin{bmatrix} \frac{\partial z_r}{\partial \mathbf{p}_f^c} \frac{\partial \mathbf{p}_f^c}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{z}_b}{\partial \mathbf{p}_f^c} \frac{\partial \mathbf{p}_f^c}{\partial \mathbf{x}} \end{bmatrix} \Big|_{\hat{\mathbf{x}}} \begin{bmatrix} \tilde{\mathbf{x}} + \boldsymbol{\eta}^r \\ \tilde{\mathbf{z}}_b + \boldsymbol{\eta}^b \end{bmatrix} =: \Lambda \begin{bmatrix} \mathbf{H}_r \\ \mathbf{H}_b \end{bmatrix} \mathbf{H}_x \tilde{\mathbf{x}} + \Lambda \begin{bmatrix} \boldsymbol{\eta}^r \\ \boldsymbol{\eta}^b \end{bmatrix} =: \mathbf{H}_x \tilde{\mathbf{x}} + \boldsymbol{\eta}^p \quad (11.43)$$

where  $\tilde{\mathbf{z}}_p$  is the measurement at the linearization point. Depending on the selection matrix  $\Lambda$ , the Jacobian  $\mathbf{H}_x$  may include the range-only measurement Jacobian  $\mathbf{H}_r$  ( $\lambda_r = 1$ ,  $\lambda_b = 0$ ), the bearing-only Jacobian  $\mathbf{H}_b$  ( $\lambda_r = 0$ ,  $\lambda_b = 1$ ), or both.

*line measurements* **Line Features.** Given two 3D points  $\mathbf{p}_1^w$  and  $\mathbf{p}_2^w$ , we can represent the line passing through the two points using its Plücker coordinates:

$$\mathbf{l}^w = \begin{bmatrix} \mathbf{n}_\ell^w \\ \mathbf{v}_\ell^w \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^w \times \mathbf{p}_2^w \\ \mathbf{p}_2^w - \mathbf{p}_1^w \end{bmatrix} \quad (11.44)$$

where  $\mathbf{n}_\ell^w$  is the line moment that encodes the normal direction of the plane defined by the two points and the origin, and  $\mathbf{v}_\ell^w$  is the line direction vector which can be normalized to a unit vector if needed. Note that the distance from the origin to the line can be computed as  $d_\ell^w = \frac{\|\mathbf{n}_\ell^w\|}{\|\mathbf{v}_\ell^w\|}$ , and the above Plücker coordinate—expressed in the world frame—can be transformed to the camera frame as [1018]:

$$\begin{bmatrix} \mathbf{n}_\ell^c \\ \mathbf{v}_\ell^c \end{bmatrix} = \begin{bmatrix} \mathbf{R}_c^{w^\top} & -\mathbf{R}_c^{w^\top}(\mathbf{p}_c^w)^\wedge \\ \mathbf{0} & \mathbf{R}_c^{w^\top} \end{bmatrix} \begin{bmatrix} \mathbf{n}_\ell^w \\ \mathbf{v}_\ell^w \end{bmatrix} \quad (11.45)$$

We now consider a case where the 3D line is observed in 2D images. Specifically, given two endpoints of a line segment in the image:  $\mathbf{q}_1 := [u_1, v_1, 1]^\top$  and  $\mathbf{q}_2 := [u_2, v_2, 1]^\top$ , we derive the 2D visual line measurement model as the distances of these two endpoints to the back-projected 3D Plücker line onto the image plane [1312]. To this end, we transform the 3D line from the world frame to the current camera frame via (11.45) and then project it onto the image with the known intrinsic parameters of the camera [1018]:

$$\boldsymbol{\ell} = \underbrace{\begin{bmatrix} f_2 & 0 & 0 \\ 0 & f_1 & 0 \\ -f_2 c_1 & -f_1 c_2 & f_1 f_2 \end{bmatrix}}_{\mathbf{K}} [\mathbf{I}_3 \quad \mathbf{0}_3] \begin{bmatrix} \mathbf{n}_\ell^c \\ \mathbf{v}_\ell^c \end{bmatrix} =: \begin{bmatrix} \ell_1 \\ \ell_2 \\ \ell_3 \end{bmatrix} \quad (11.46)$$

where  $\mathbf{K}$  is the canonical projection Plücker matrix and  $f_1, f_2, c_1$  and  $c_2$  are the standard camera intrinsic parameters. Note that only the moment vector  $\mathbf{n}_\ell^c$  in the Plücker coordinates is involved in the above projection, which implies that the line range and orientation contained in  $\mathbf{v}_\ell^c$  are not measurable. Therefore, the distances of the two endpoints of the line segment to the projected line  $\boldsymbol{\ell}$  in the image can be finally computed and used as the line feature measurements:

$$\mathbf{z}_\ell = \begin{bmatrix} \frac{\mathbf{q}_1^\top \boldsymbol{\ell}}{\sqrt{\ell_1^2 + \ell_2^2}} \\ \frac{\mathbf{q}_2^\top \boldsymbol{\ell}}{\sqrt{\ell_1^2 + \ell_2^2}} \end{bmatrix} + \boldsymbol{\eta}^\ell \quad (11.47)$$

where  $\boldsymbol{\eta}^\ell$  is the measurement noise. Similarly, with the chain rule of differentiation, we can linearize (11.47) with respect to the state and obtain the measurement Jacobian:  $\mathbf{H}_x = \frac{\partial \mathbf{z}_\ell}{\partial \boldsymbol{\ell}} \frac{\partial \boldsymbol{\ell}}{\partial \mathbf{x}}$ .

**Plane Features.** A 3D plane can be parameterized by its distance to the origin and normal direction in the world frame:  $\boldsymbol{\pi}^w = \begin{bmatrix} \mathbf{n}_\pi^w \\ d_\pi^w \end{bmatrix}$ , which can be transformed to the local sensor frame where the plane feature is typically detected:

$$\begin{bmatrix} \mathbf{n}_\pi^c \\ d_\pi^c \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^c & \mathbf{0}_{3 \times 1} \\ -(\mathbf{p}_c^w)^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{n}_\pi^w \\ d_\pi^w \end{bmatrix} \quad (11.48)$$

Without loss of generality, we consider a plane feature  $(\mathbf{n}_\pi^c, d_\pi^c)$  is extracted from point clouds (*e.g.*, LiDAR or depth sensors), and employ the closes point  $\mathbf{p}_\pi^c =$

*plane measurements*

$d_\pi^c \mathbf{n}_\pi^c$  from the plane to the origin as the plane representation in the AINS state vector [374].

$$\mathbf{z}_\pi = d_\pi^c \mathbf{n}_\pi^c + \boldsymbol{\eta}^\pi = \mathbf{p}_\pi^c + \boldsymbol{\eta}^\pi \quad (11.49)$$

where  $\boldsymbol{\eta}^\pi$  is the plane measurement noise. Linearization of (11.49) yields the plane measurement Jacobian  $\mathbf{H}_x = \frac{\partial \mathbf{z}_\pi}{\partial \mathbf{p}_\pi^c} \frac{\partial \mathbf{p}_\pi^c}{\partial \mathbf{x}}$ .

### 11.3.2 Observability Analysis

Based on the linearized system and measurement models presented in the previous sections, we can now perform the observability analysis. The analysis relies on the following *observability matrix*  $\mathbf{M}(\hat{\mathbf{x}})$  to gain insights about the system (*cf.* [486]):

$$\mathbf{M}(\hat{\mathbf{x}}) = \begin{bmatrix} \mathbf{H}_{x_1} \Phi_{(1,1)} \\ \mathbf{H}_{x_2} \Phi_{(2,1)} \\ \vdots \\ \mathbf{H}_{x_k} \Phi_{(k,1)} \end{bmatrix} \quad (11.50)$$

where  $\mathbf{H}_{x_k}$  stacks the Jacobians for all the measurements (of points, lines, or planes) collected at discrete time  $k$ , and the notation  $\mathbf{M}(\hat{\mathbf{x}})$  stresses the fact that the observability matrix depends on the linearization point  $\hat{\mathbf{x}}$ . The nullspace  $\mathcal{U}$  of this matrix, *i.e.*, the span of the null vectors  $\text{span}([\dots \mathbf{u}_i \dots]) = \mathcal{U}$  such that  $\mathbf{M}(\mathbf{x})\mathbf{u}_i = \mathbf{0}$ , describes the unobservable subspace of AINS. If the nullspace is empty, the system is fully observable. It has been shown in [1231] that AINS in general has 4 unobservable directions (*i.e.*, it has four independent vectors in the null space  $\mathcal{U}$ ), describing the fact that the global 3D position and global yaw are unobservable from IMU measurements and local observations of previously unknown landmarks.

To understand the structure of the 4-dimensional null space, we consider the case where all three types of geometric features (*i.e.*, a single point, line, and plane) are in the state vector:  $\mathbf{x}_f^w = \{\mathbf{p}_f^w, \mathbf{l}^w, \boldsymbol{\pi}^w\}$ , and the exteroceptive measurements include:  $\mathbf{z} = \{\mathbf{z}_p, \mathbf{z}_\ell, \mathbf{z}_\pi\}$  (*cf.* (11.42), (11.47), and (11.49)). By computing the related system and measurement Jacobians (*i.e.*,  $\mathbf{H}_{x_i}$  and  $\Phi_{(i,1)}$ ) and substituting them into (11.50), we can build the corresponding linearized AINS observability matrix  $\mathbf{M}$ . By mathematically computing the nullspace of this matrix  $\text{null}(\mathbf{M})$ , we should be able to find the following four null-vectors (*cf.* [1231]):

$$\text{null}(\mathbf{M}) = \text{span}[\mathbf{u}_1 \ \mathbf{u}_{2:4}] = \text{span} \begin{bmatrix} \mathbf{u}_g & \mathbf{0}_{12 \times 3} \\ -\mathbf{p}_1^w \times \mathbf{g}^w & \mathbf{I}_3 \\ -\mathbf{p}_f^w \times \mathbf{g}^w & \mathbf{I}_3 \\ -\mathbf{g}^w & \frac{\mathbf{v}_\ell^w}{d_\ell^w \|\mathbf{v}_\ell^w\|} (\mathbf{R}_\ell^w \mathbf{e}_1)^\top \\ 0 & -(\mathbf{R}_\ell^w \mathbf{e}_3)^\top \\ -d_\pi^w \mathbf{n}_\pi^w \times \mathbf{g}^w & \mathbf{n}_\pi^w (\mathbf{R}_\pi^w \mathbf{e}_3)^\top \end{bmatrix} \quad (11.51)$$

where  $\mathbf{u}_g = [(\mathbf{R}_w^{c_1} \mathbf{g}^w)^\top \quad \mathbf{0}_{1 \times 3} \quad -(\mathbf{v}_1^w \times \mathbf{g}^w)^\top \quad \mathbf{0}_{1 \times 3}]^\top$ ,  $\mathbf{p}_1^w$  refers to the sensor position at the time  $k = 1$ ,  $\mathbf{R}_w^{c_1}$  is the rotation matrix from the sensor frame  $C_1$  at time  $k = 1$  to the world frame  $W$ , while  $\mathbf{R}_\pi^w$  is a rotation matrix built using the plane normal vector  $\mathbf{n}_\pi^w$  using Gram–Schmidt orthonormalization (*cf.* (11.6)), and  $\mathbf{R}_\ell^w = \left[ \frac{\mathbf{n}_\ell^w}{\|\mathbf{n}_\ell^w\|} \quad \frac{\mathbf{v}_\ell^w}{\|\mathbf{v}_\ell^w\|} \quad \frac{\mathbf{n}_\ell^w}{\|\mathbf{n}_\ell^w\|} \times \frac{\mathbf{v}_\ell^w}{\|\mathbf{v}_\ell^w\|} \right]$  is the rotation matrix constructed with the line normal and line direction. It is possible to see that the first null vector  $\mathbf{u}_1$  is related to the rotation around the gravity (and hence the yaw) and  $\mathbf{u}_{2:4}$  to the motion of the robot. Readers are referred to [1231, 1230] for more detailed analysis.

In summary, the fact that the observability matrix admits a 4-dimensional null space (*cf.* (11.51)) correctly describes the fact that the global position and yaw of the system are not observable. Intuitively, none of the measurements (IMU data, measurements of unknown point, line, or plane landmarks) convey information about the global frame, with the exception of roll and pitch, which are observable from the accelerometer measurements of the gravity direction. This unobservability is common in SLAM<sup>9</sup> and it is not pathological: it only means that we can arbitrarily set the yaw and 3D origin of our world frame since we only have relative measurements for those variables. This unobservability would disappear when adding a sensor providing absolute measurements, *e.g.*, a GPS. More concerning is that fact that for certain motions (and linearization points), the null space of the observability matrix can grow larger, creating additional unobservable dimensions. We explore this phenomenon below.

*unobservable directions*

### 11.3.3 Degenerate Motions

Certain types of motions might induce additional unobservable directions for AINS (*i.e.*, in addition to the 4 expected ones that we discussed above). This is of practical importance since these degenerate motions might lead to large errors in some directions of the state space and lead to navigation failures. The degenerate motions of AINS are summarized in Table 11.1 (see [1231] for a full derivation). Specifically, pure translation is degenerate for all feature types, causing the full global rotation to become unobservable. Intuitively, if the system is not rotating, we might confuse gravity measurements with accelerometer biases, hence making the roll and pitch no longer observable. The other three degenerate motions, namely constant acceleration (including the case of constant velocity, where the acceleration is set to zero), pure rotation, and motion in the direction of the feature (for the case where we have a single point feature), cause the scale to be unobservable for the case of monocular camera (*i.e.*, bearing-only measurements). However, constant acceleration causes the whole system (*i.e.*, position, velocity, acceleration bias, and features) scale to be unobservable, while pure rotation and moving toward a feature only make the

<sup>9</sup> Without using an IMU, the null space for a landmark-based SLAM problem would be at least 6-dimensional, capturing the fact that without an IMU, the entire 3D rotation (in addition to the 3D position) of the system is unobservable.

feature scale unobservable. Note that these three degenerate motions hold only if the distance from the sensor to the feature is significantly larger than the extrinsic translation between the sensor and robot body (if they do not coincide), *i.e.*,  $\|\mathbf{p}_f^c\| \gg \|\mathbf{p}_b^c\|$ , which typically is the case in practice.

Table 11.1 *Degenerate motions of AINS.*

<i>Motion</i>	<i>Sensor</i>	<i>Unobservable</i>
1. Pure translation	General	Global orientation
2. Constant acceleration	Mono cam	System scale
3. Pure rotation	Mono cam	Feature scale
4. Moving toward point feature	Mono cam	Feature scale

## 11.4 Visual-Inertial Odometry and Practical Considerations

As mentioned above, inertial measurements are typically fused with data from other sensors to mitigate the odometry drift. In this section, we particularly focus on the case where visual measurements from a camera are fused with IMU measurements using factor graphs.<sup>10</sup> Camera and IMU are a popular combination, since they are both inexpensive, lightweight, and low-power sensors. Moreover, they are complementary sensors, where the IMU is able to capture quick acceleration and rotations, while cameras are able to provide rich observations of the surrounding environment. On one side, the use of cameras largely reduces the drift as compared to pure inertial odometry; on the other side, an IMU might allow observing quantities that would not be possible to estimate otherwise. In particular, when using a monocular camera for SLAM, one cannot estimate the scale of the scene without relying on prior information (in other words, the scale is unobservable), while adding an IMU allows retrieving the scale as well, as long as the motion of the robot is non-degenerate (Section 11.3.3). As we mentioned earlier in this chapter, systems comprising one or more cameras as well as an IMU are typically referred to as visual-inertial odometry (VIO) systems, and become visual-inertial SLAM systems when loop closures are incorporated.

### 11.4.1 Visual-Inertial Odometry

VIO systems are commonly used as a source of odometry and are often used to close control loops over trajectory tracking and control. In other applications, such as virtual reality, VIO systems are used to compensate for the motion of the user in the virtual environment. In both cases, VIO is required to produce estimates with

<sup>10</sup> In robotics, it is also common to use inertial data in combination with other sensors, including LiDAR and radars, see Chapter 8 and Chapter 9.

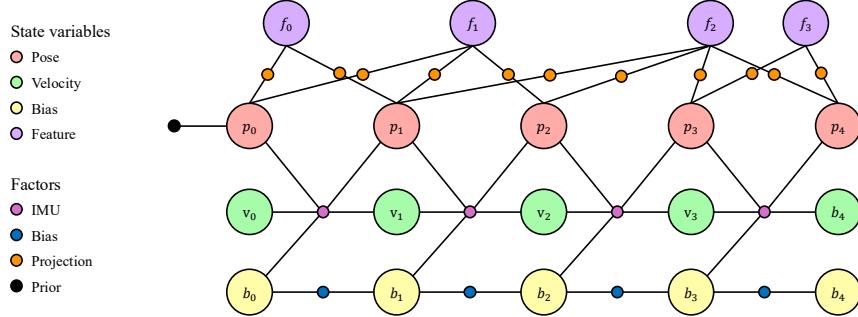


Figure 11.4 Example of factor graph used for visual-inertial odometry with preintegrated IMU factors [335]. The factor graph shows preintegrated IMU factors in violet (constraining consecutive poses, velocity, and bias), bias factors in blue (constraining the evolution of the IMU biases over time), vision factors in orange (relating camera poses and positions of external landmarks), and priors in black.

very low-latency, typically in the order of 10-50ms. For instance, the refresh rate of the Meta Quest 3 is between 72Hz and 120Hz [758], and the VIO latency directly impacts the quality of the VR experience and is key to mitigating motion sickness. Similarly, for trajectory tracking it is important to keep the latency low since large delays might induce instability and divergence of the tracking controller.

Based on these considerations, factor-graph-based VIO systems typically implement a fixed-lag smoother (also called sliding-window optimization), where one only attempts to estimate states in a receding horizon (*e.g.*, the last 5-10 seconds). An example of the resulting factor graph is shown in Fig. 11.4, which shows preintegrated IMU factors in violet, bias factors in blue, vision factors in orange, and priors in black. The horizon is chosen in a way to trade-off computation with accuracy, since the longer the horizon, the larger the state space for estimation. Then factors and variables falling out of the receding horizon are gradually marginalized as time progresses. In many cases, optimized implementations also eliminate visual landmarks from the optimization using the Schur complement to further reduce the size of the state space, see, *e.g.*, [335]. An alternative to using a fixed-lag smoother is to use an incremental solver like iSAM2 (Section 1.7), which reuses computation from previous optimizations when computing an estimate at the current time. While this approach has been shown to lead to very accurate results in practice [335], it has the drawback of not providing guarantees on the latency of the system and might lead to spikes in the runtime, which is problematic for certain applications.

sliding-window optimization

**VIO Systems and Performance.** The last decade has seen a proliferation of visual-inertial odometry/SLAM systems, many of which have open-source implementations. Popular approaches include a visual-inertial version of ORB-SLAM [783], Direct Sparse Visual-Inertial Odometry [1113], VINS-Mono [893], OpenVINS [375], Kimera [3, 939], BASALT [1115], and DM-VIO [1040]. A good VIO system has a

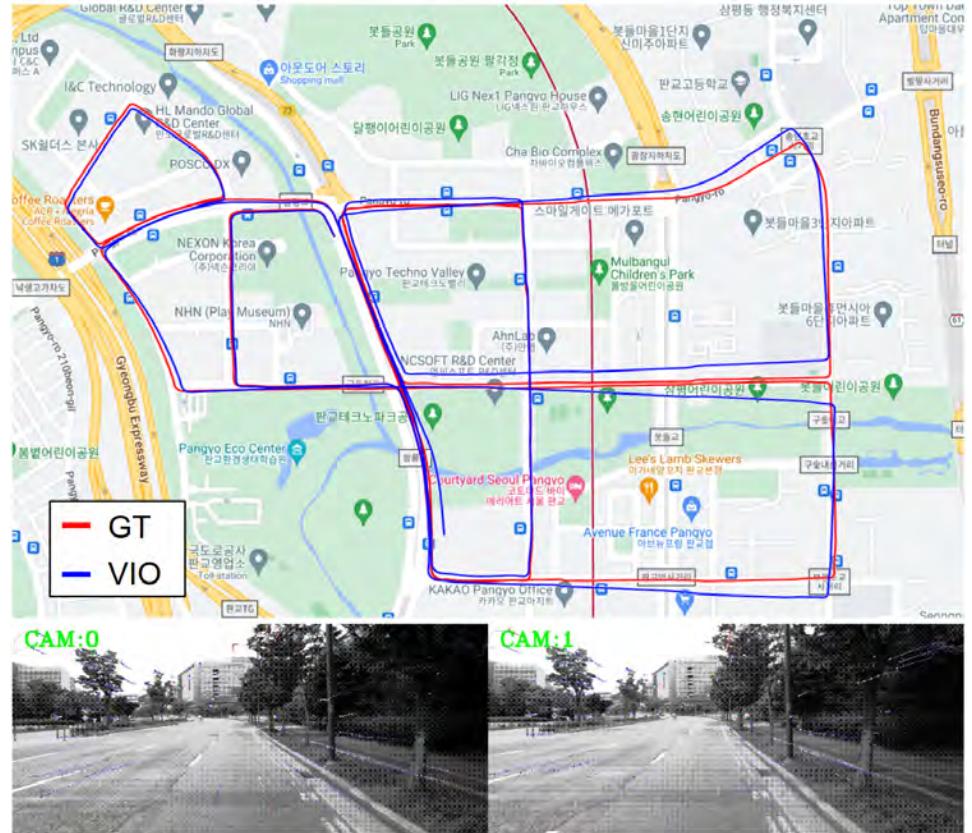


Figure 11.5 Illustration of a recent sliding-window optimization-based VIO algorithm [180] running on the KAIST urban autonomous driving dataset, sequence 38. This sequence has a total duration of 36 minutes and is 11.42 km in length. The VIO estimates and ground truth are overlaid on the Google map. Bottom are two sample images. The final ATE of the VIO (i.e., without loop closure) is 2.05 degrees and 21.2 meters (0.18%).

drift below 1% of the distance traveled (*e.g.*, it accumulates an error smaller than 1m after covering a 100m trajectory), and in some cases the drift can be as low as 0.1%.

Sliding-window optimization approaches such as VINS-Mono [893] have seen tremendous success in practice. As an example, here we show how a more recent sliding-window-based approach, called First-Estimate Jacobian (FEJ)-based Window Bundle Adjustment (WBA)-VINS [180, 181] performs on the KAIST Urban Dataset [514]. The KAIST urban dataset focuses on autonomous driving and localization in challenging complex urban environments. The dataset was collected in Korea with a vehicle equipped with stereo camera pair, 2D/3D LiDARs, Xsens IMU, Fiber Optic Gyro (FoG), wheel encoders, and RKT GPS. The camera oper-

ates at 10Hz, while the IMU sensing rate is 100Hz. A ground-truth trajectory is also provided which is obtained from the fusion of the FoG, RKT GPS, and wheel encoders. Fig. 11.5 illustrates the FEJ-WBA-VINS [180, 181] (VIO) estimated trajectory for *sequence 38*, overlaid on a Google map alongside the ground truth (GT) trajectory. The final absolute trajectory error (ATE) for the 11.42-kilometer path is approximately 2.05 degrees and 21.2 meters (0.18% of the trajectory traveled). Notably, these results are obtained from the pure online VIO without loop closure.

Applications of VIO to self-driving cars and other autonomous systems is discussed in [3, 4], which also discusses challenges related to feature tracking, keyframe selection, and fusion of different sensor modalities (including monocular, stereo, and RGB-D camera images, as well as wheel odometry).

#### 11.4.2 Extrinsic Calibration

In order to enable accurate aided inertial navigation, one needs to perform extrinsic calibration of the sensors. The extrinsic calibration corresponds to estimating the relative pose between the different sensors (*e.g.*, the pose of the camera with respect to the IMU). The literature provides a variety of methods to address the calibration problem. These can be mainly divided into two categories: offline and online calibration methods. Offline approaches require a calibration procedure to be performed before the system is deployed. It often involves the use of a calibration target [350], a known motion pattern [683], or prior knowledge about the environment [629, 711]. These procedures can be more-or-less time consuming, and may require specific equipment and trained operators. Thus, while generally more accurate, offline calibration methods can be cumbersome and undesirable in some scenarios where the final system is expected to be used at large scale by non-experts. Online methods, on the other hand, do not require a specific procedure [301, 631, 1208, 1234]. Instead, the extrinsic calibration parameters are estimated as part of the state estimation problem. This offers the advantage of being able to adapt to changes in the system, such as sensor displacement, without the need for a new calibration procedure. However, online calibration methods can be less accurate than offline methods, and may render the state estimation problem more complex or even ill-posed [1232].

#### 11.4.3 Temporal Synchronization

Another crucial aspect of inertial-aided system is the temporal synchronization of the sensor data. An erroneous synchronization that is not accounted for can lead to significant errors in the trajectory estimates and/or introduce a bias in the benchmarking metrics. The synchronization can be done either in hardware or in software. The low-level hardware approach often relies on a dedicated piece of hardware that triggers the various sensors' data acquisition based on a common clock signal via specific synchronization input pins. This is not always possible, especially when

*calibration*

*temporal synchronization*

the sensors are connected to the computer via different communication protocols. Some sensors may have a built-in synchronization mechanism that can be used to synchronize the different sensors' clock without the need for a dedicated hardware input. The use of PTP (Precision Time Protocol) is an example of such a software-based synchronization over Ethernet. Many LiDARs, radars and INS solutions can be synchronized with this protocol. Another solution is time-stamping the data at the sensor level and then aligning the timestamps in a post-processing step. This last approach is generally less accurate and robust than the aforementioned ones. If the system cannot be synchronized and post-processing is not an option (e.g., online applications), some state estimation algorithms integrate the time offset as a state variable in the estimation problem [301, 376, 1234].

### 11.5 Further Readings & Recent Trends

While progress in inertial odometry is steadily transitioning into industry products, aided inertial navigation is still the subject of intense research.

**Extended Pose Preintegration.** Latest trends in inertial odometry for SLAM include the use of extended-pose manifolds and higher-order noise propagation [120] to improve the uncertainty modeling of IMU preintegration. Brossard et al. [120] extend the preintegration theory to account for Earth's rotation with the Coriolis and centrifugal forces. Vial et al. [1123] provide an example of extended pose preintegration leveraging both a linear velocity sensor and a navigation-grade IMU. After an hour of marine navigation over a 1.8km trajectory, the authors report a translation error of around 5m.

**Continuous-time State Representations.** We have mostly been interested in the use of IMUs under the scope of preintegration as a mean to reduce the number of discrete state variables in our factor graphs. However, other approaches bases on continuous-time state representation can also account from many IMU measurements without increasing the dimensionality of the estimated state. We find such examples in [349] using B-splines basis functions and in [47] with GP priors. Both formulations allow to use IMU measurements at high rates in residuals based on interpolated dynamics between a fixed set of state variables. Recently, the work [130] compares the integration of IMU measurements directly as inputs in the continuous-time GP prior against using IMU measurements directly in residuals. They concluded that using inertial information as measurements of the state resulted in better odometry accuracy using a LiDAR-inertial sensor suite. Another interesting work on continuous-time representations is [657], where the authors compare the GP-based state representation from [47] with the continuous GP-based preintegration from [376] (presented earlier in this chapter). In a event-based VIO context, the authors show that the later provides a slight advantage over the former both in terms of accuracy and computational efficiency.

**Proprioception-only Odometry.** Recent works use proprioceptive sensors for aided inertial navigation. For odometry, works like [441] with legged robots and [810] with wheel-mounted IMUs demonstrate how some knowledge about the system’s kinematics can be used to provide competitive IMU-based odometry estimates with sub-percent positional error. In [441] the critical information is the knowledge of contact between the robot’s feet and the ground, while in [810] the one-plane-rotation motion is used to constrain the IMU biases and therefore limit the dead-reckoning drift. The work [810] has been extended into a full SLAM system [1197] by detecting loop closures based on pattern recognition in the road bank angle over time, providing an interesting example of an IMU-based proprioceptive system that can perform loop closure detection and correction. It is important to note that while the use of inertial sensors generally offers better performance and robustness, dropouts or saturation of the IMU sensor can have catastrophic effects on the overall system’s performance. In [266] the authors investigate the use of accelerometer data to estimate angular velocity when the gyroscope saturates, thus improving the robustness of downstream SLAM algorithms.

**Inertial-only Odometry (IOO).** Naive integration of IMU measurements — without aiding sources such as vision— typically leads to quick divergence of the odometric estimate. This is a cause of concern even in aided inertial odometry when the source of aiding becomes unavailable. For example, for hand tracking in mobile AR/VR applications, highly dynamic hands can easily move out of the tracking camera’s FOV, leaving only IMU data available to keep motion tracking alive; or textureless scenes may prevent feature detection and tracking, causing VIO to only rely on IMU data. For this reason, recent work investigates the use of learning and neural networks to reduce the drift in inertial-only odometry [1212, 179, 1052, 451, 452, 220, 895]. These include attempts to model IMU bias in a data-driven manner with neural networks [225] or directly predicting displacements from a sequence of noisy IMU measurements [679]. For instance, one may use a differentiable integration module to integrate IMU readings with the predicted bias removed [1273, 895], or directly use ground truth bias for supervision [123], or use a conditional diffusion model to approximate bias which is modeled as a probability distribution [1295]. These methods have demonstrated the possibility of largely reducing the drift in inertial-only odometry, but currently they have limited generalization (*e.g.*, to different sensors or to motions not seen at training time).

**Ultra-efficient and Robust VIO at the Edge.** Despite recent advancements in SLAM, computational constraints arising in embedded robotic systems still pose critical challenges. Building robust VIO on these small form-factor platforms is hard due to strict size, weight, and power (SWAP) constraints, with the primary difficulty often arising from data management rather than computation. For example, in SLAM and hand-tracking modules of Meta XR wearable devices, the major energy consumption is data access in RAM [621]. To reduce the data transfer, an

on-sensor computing architecture is presented [390], and a quantized visual-inertial odometry (QVIO) algorithms is developed in [862, 864]. For low-SWaP platforms, where only single-precision floating-point arithmetic is available on the computation unit or is required to speed up and achieve real-time performance, new square-root (information or covariance) filters [863, 1191] have been introduced to improve efficiency while maintaining numerical stability. An ASIC design and implementation for on-chip visual-inertial odometry system is presented in [1276, 1045].

# 12

## Leg Odometry for SLAM

Marco Camurri and Matías Mattamala

Legged robots are becoming widespread thanks to their ability to traverse highly unstructured terrains. Their main advantage is that legs provide an active suspension that decouples the motion of the robot's body from the terrain profile [985]. This enables them to negotiate staircases, uneven terrain, and other ground obstacles that are challenging for wheeled platforms [218, 500]. Even though the SLAM algorithms reviewed in the other chapters are directly applicable to legged robots, the additional sensing introduced by the legs is a valuable new source of information that can be exploited for odometry. This is particularly important for legged locomotion control and planning, where high-frequency and low drift real-time pose and velocity estimation is required to prevent falls and failures, which is challenging.

In this chapter, we discuss the foundations of estimating the real-time pose and velocity of a legged robot equipped with an onboard IMU and joint sensing (position and torque). We will particularly focus on *leg odometry*, which aims to determine the relative motion of the robot's body from leg sensing. We start by providing some historical background and preliminaries (Section 12.1). Then we introduce the key theoretical tools used to estimate leg odometry (Section 12.2 and Section 12.3). In particular, we explain how the relative motion can be obtained from the joint sensing of the legs, assuming that the stance legs are known (Section 12.2). This practically allows us to consider leg odometry as a *measurement* in our SLAM problem, in a similar way to wheel odometry or inertial preintegration. Then, we describe how contact is estimated, and the main techniques adopted in practice (Section 12.3). Afterwards, we discuss how to combine leg odometry with different sensor modalities using factor graphs (Section 12.4). We conclude the chapter with a review of the open problems in the field (Section 12.5), as well as a discussion of new trends that have arisen to address them (Section 12.6).

### 12.1 Historical Background and Preliminaries

While the majority of modern legged platforms carry sensors we have studied in other chapters, *e.g.*, cameras (Chapter 7), LiDAR (Chapter 8), and inertial measurement units (Chapter 11), in legged platforms we can additionally exploit kine-

matic and dynamic information from the robot’s legs to obtain measurements of the relative motion of the robot’s body, a technique known as *leg odometry*. The term was introduced in analogy with the odometry of wheeled vehicles, which infer the distance traveled by measuring how much the wheels turn over time.

In contrast to wheeled vehicles, legged robots move by making and breaking contacts between their legs and the ground. Each leg stride is defined by a phase where the leg is temporarily lifted off the ground (*aerial* or *swing phase*), and then it stays in non-slipping contact with the ground (*stance phase*). Because the legs in the stance phase are the ones responsible for propelling the robot, the leg odometry problem can be decomposed into two sub-problems:

- 1 *Contact estimation*—establishing what legs are in stance phase at a given period of time.
- 2 *Motion estimation*—determining the incremental motion from such legs during the stance phase.

Before delving into leg odometry, we briefly review historical background and preliminary concepts in this section.

### **12.1.1 Historical Background**

Leg odometry has been directly related to the development of machines able to walk. The origins of legged robotics go back to 1950s and 1960s, with the first attempts to create transportation systems able to overcome the limitations of wheeled vehicles on rough terrain [671]. General Electric’s *Walking Truck* [773], a 1400 kg machine, is one of the best examples of the human-operated machines designed in this period. With the development of new control strategies in the 1960s, the efforts shifted to smaller scale platforms that could generate automatic walking behavior [747, 748, 1097]. Raibert’s monopods, bipeds, and quadrupeds developed in the late 1980s showed remarkable locomotion capabilities [899], motivating the use of legged platforms for real-world tasks. Achieving real-world autonomy has then been the main driver to develop leg odometry and state estimation systems.

The work by Roston and Krotkov [944] presented one of the earliest uses of leg kinematics for the estimation of the motion of a legged platform—the Ambler hexapod, a 2.5 tonnes robot designed for space exploration. Similar approaches were developed for other hexapods with different leg morphologies, such as RHex [666]. Later works combined leg odometry with other sensor modalities such as IMU and vision, via particle filters [208] or Kalman filters [202, 923]. The milestone work by Bloesch et al. [88] established the foundations for filtering-based, all terrain, kinematic-inertial odometry estimators for quadrupeds [89], and paved the way for extensions to bipedal platforms [945, 871].

The DARPA Robotics Challenge (DRC), developed between 2012 and 2015, motivated the development of *whole-body* state estimation systems for humanoid robots,

aiming not only to estimate a 6-DoF pose but also the full state of the robot’s body. For this, various teams combined kinematic and dynamic information [1200] with inertial and joint sensing, as well as exteroceptive modalities such as LiDAR [315] and stereo vision [316]. In this period, most of the solutions relied on Kalman filtering but a few works also explored optimization-based approaches [1201] and factor graphs [336].

With the recent rise and commercialization of legged platforms, particularly quadrupeds, there has been a growing interest in developing more principled and resilient leg odometry and state estimation algorithms. This has been reflected in the further development of factor-graph-based estimation solutions for quadrupedal [1188] and bipedal platforms [439], which have enabled its principled fusion with other sensor modalities. Other directions have explored more fundamental challenges in modelling [19], as well as invariant estimation frameworks [441, 438]. The recent DARPA Subterranean (SubT) Challenge (2018-2021) also posed new challenges for legged state estimation in extreme scenarios, where *complementary* estimation solutions leveraged different sensor modalities across diverse legged, wheeled, and aerial platforms [557, 1284, 835]. Section 12.6 will provide further insights on the recent trends, and how they are re-shaping the research in state estimation for legged platforms.

### 12.1.2 Reference Frames

Figure 12.1 illustrates the reference frames relevant to our estimation problem. The inertial frame  $\mathcal{F}^w$  and the base frame  $\mathcal{F}^b$  are rigidly attached to the ground and the robot’s floating base, respectively. Without loss of generality, we assume the IMU to be coincident with  $\mathcal{F}^b$ . A frame is attached to each end effector, corresponding to the feet ( $\mathcal{F}^{f1}$  and  $\mathcal{F}^{f2}$  in the example shown in Figure 12.1).

Additionally, one or more temporary inertial frames  $\mathcal{F}^k$  are created when a foot comes into contact with the ground. These are coincident with the foot frame at touchdown for humanoids, or have the same Cartesian position for quadrupeds with point feet.

### 12.1.3 State Definition

The state of a legged robot is defined by the pose and velocity of its base, as well as the joint states. However, in this chapter we assume the joint states to be measured directly by dedicated sensors (Section 12.1.6), leaving only the pose and velocity of the robot’s base as the objective of our estimation. Therefore, we use the term *state estimation* and odometry interchangeably, with the latter term being equivalent to SLAM without loop closures [136].

More formally, the robot state is defined as the set combining position, orienta-

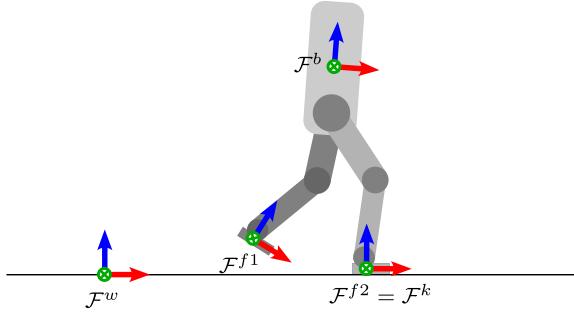


Figure 12.1 Reference frame conventions for legged robots. The world frame  $\mathcal{F}^w$  is fixed to earth, while the base frame  $\mathcal{F}^b$  is attached to the main chassis. Without loss of generality, the IMU frame is not shown as it can be considered coincident with  $\mathcal{F}^b$ . When a foot touches the ground, a contact frame  $\mathcal{F}^k$  is defined.  $\mathcal{F}^k$  is rigidly attached to earth, perpendicular to the ground, and coincident with the foot frame  $\mathcal{F}^{f2}$ .

tion, linear velocity, and angular velocity:

$$\boldsymbol{x}_k = [\boldsymbol{t}_k \quad \boldsymbol{R}_k \quad \boldsymbol{v}_k \quad \boldsymbol{\omega}_k]^\top \quad (12.1)$$

where the following conventions are adopted: the robot position  $\boldsymbol{t} = \boldsymbol{t}_b^w \in \mathbb{R}^3$  and orientation  $\boldsymbol{R} = \boldsymbol{R}_b^w \in \text{SO}(3)$  express the pose of the robot's base in world coordinates; the robot velocities  $\boldsymbol{v} = \boldsymbol{v}_b^b$ ,  $\boldsymbol{\omega} = \boldsymbol{\omega}_b^b \in \mathbb{R}^3$  express the base's twist, in base coordinates.

#### 12.1.4 Legged Robot Kinematics

A legged robot is kinematically described by a main link for the body (also called trunk, or torso) to which one or more kinematic chains (*i.e.*, the legs) are attached. In this chapter, we consider the most common kinematic configurations adopted in practice: bipeds with 6 actuated DoF per leg and flat feet, and quadrupeds with 3 DoF per each leg and point feet (Figure 12.2). We assume that the robot has a rigid body (*e.g.*, with no articulated spine), and ignore any other upper limbs (*e.g.*, arms).

For leg odometry, we are interested in modeling the relative pose (or position) of the flat (or point) feet with respect to the robot's body. Let  $\boldsymbol{q} \in \mathbb{R}^N = [q_1, \dots, q_N]^\top$  be the set of joint positions of an articulated robot with  $N$  active DoFs, which correspond to the angular position of revolute joints of the legs. In Figure 12.1 and for the rest of the chapter, the number of active DoFs of the quadruped and biped is  $N = 12$ , although this can be different in other platforms. The joint positions  $\boldsymbol{q}$  are typically measured directly via rotary encoders placed on each joint (see Section 12.1.6.1). Alternatively, they can be measured indirectly using the kinematic model of the robot and the readings from encoders placed on a transmission between the

*joint positions*

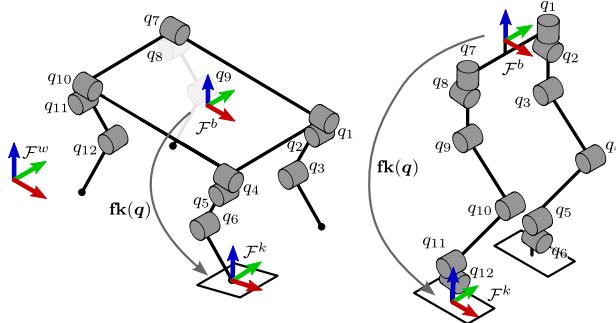


Figure 12.2 Kinematic chains of typical legged robots: quadrupeds and humanoids.

motor and the joint (*e.g.*, by measuring the displacement of a hydraulic piston via a linear encoder, and calculating the corresponding angle of the revolute joint moved by the piston).

The time derivatives of the joint positions are the joint velocities  $\dot{\mathbf{q}} = [\dot{q}_1, \dots, \dot{q}_N]^T$ , *joint velocities* which are typically estimated by numerical differentiation of the encoder readings. In some cases, when the reading are particularly noisy, additional sensing such as IMUs placed at the links can also be used to estimate the joint velocities [1202].

Given  $\mathbf{q}$ , for each foot  $f$  we define the corresponding *forward kinematics* function *forward kinematics*  $\mathbf{fk}(\mathbf{q}) : \mathbb{R}^{12} \rightarrow \text{SE}(3)$  [714], which maps the joint positions to the pose of the foot with respect to the robot's base:

$$\mathbf{T}_f^b = \mathbf{fk}(\mathbf{q}) = \begin{bmatrix} \mathbf{f}_R(\mathbf{q}) & \mathbf{f}_p(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix} \quad (12.2)$$

where we also define two convenient functions  $\mathbf{f}_R : \mathbb{R}^{12} \rightarrow \text{SO}(3)$  and  $\mathbf{f}_p : \mathbb{R}^{12} \rightarrow \mathbb{R}^3$  expressing the orientation and Cartesian position of the foot in the base frame, respectively. For quadruped robots with point feet, only  $\mathbf{f}_p(\mathbf{q})$  is used for leg odometry, since the foot can pivot on the contact point with no change in the joint positions.

The time derivative of the forward kinematics function of foot  $f$  from (12.2) is the *Jacobian matrix*  $\mathbf{J}(\mathbf{q}) : \mathbb{R}^{12} \rightarrow \mathbb{R}^{6 \times 12}$  which can be used to compute the velocity of a single foot with respect to the robot's base as follows [715]: *Jacobian matrix*

$$\begin{bmatrix} \mathbf{v}_f^b \\ \boldsymbol{\omega}_f^b \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} \mathbf{J}_v(\mathbf{q}) \\ \mathbf{J}_\omega(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} \quad (12.3)$$

where  $\mathbf{J}_v(\mathbf{q}) \in \mathbb{R}^{3 \times 12}$  and  $\mathbf{J}_\omega(\mathbf{q}) \in \mathbb{R}^{3 \times 12}$  are the linear and angular part of the Jacobian, respectively (note that some references [319][441] define the angular block first). Since  $\mathbf{q}$  expresses the position of all  $N = 12$  joints, but each leg is kinematically independent from the others,  $\mathbf{J}(\mathbf{q})$  is as a sparse block matrix, where we indicate the non-zero block as  $\bar{\mathbf{J}}(\mathbf{q})$ . These can be used to map the subset of joint angle velocities of a specific leg to the linear and angular velocity of the corre-

sponding foot  $f$ . For example, the Jacobian of the second leg of a humanoid  $\mathbf{J}_2(\mathbf{q})$  is represented as:

$$\mathbf{J}_2(\mathbf{q}) = [\mathbf{0}_6 \quad \bar{\mathbf{J}}_2(\mathbf{q})] = \begin{bmatrix} \mathbf{J}_{2,v}(\mathbf{q}) \\ \mathbf{J}_{2,\omega}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3 \times 6} & \bar{\mathbf{J}}_{2,v}(\mathbf{q}) \\ \mathbf{0}_{3 \times 6} & \bar{\mathbf{J}}_{2,\omega}(\mathbf{q}) \end{bmatrix} \quad (12.4)$$

where  $\bar{\mathbf{J}}_2(\mathbf{q}) \in \mathbb{R}^{6 \times 6}$  indicates the non-zero block of the Jacobian matrix.

The expressions from (12.2) to (12.3) are the basis to design state estimators for legged platforms, as they relate the joint states to the robot's base motion. However, as mentioned previously, we assume we can measure the joint states directly via encoders, as explained in Section 12.1.6.

### 12.1.5 Legged Robot Dynamics

The dynamics of a floating-base articulated-body system can be expressed as two coupled dynamics equations, computed using Recursive Newton-Euler algorithms [319]. The first equation describes the dynamics of the floating-base body (6 DoF, unactuated), while the second describes the dynamics of the  $N$  rigid-bodies (*i.e.*,  $N = 12$ ) attached to it through active joints (*i.e.*, active DoF). The two equations of motion can be put in matrix form as follows:

$$\mathbf{M}(\mathbf{q}) \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\boldsymbol{\omega}} \\ \ddot{\mathbf{q}} \end{bmatrix} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} \mathbf{J}_b^\top \\ \mathbf{J}_q^\top \end{bmatrix} \mathbf{f} + \begin{bmatrix} \mathbf{0}_6 \\ \boldsymbol{\tau} \end{bmatrix} \quad (12.5)$$

where the first term  $\mathbf{M}(\mathbf{q})$  is the mass matrix, which is multiplied by the stack of  $\dot{\mathbf{v}} \in \mathbb{R}^3$ ,  $\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$ , and  $\ddot{\mathbf{q}} \in \mathbb{R}^{12}$  which represent the floating-base linear, floating-base angular, and active joints accelerations, respectively. The second term  $\mathbf{h} \in \mathbb{R}^{18}$  is a bias term that accounts for Coriolis, centrifugal, and gravitational effects. Regarding the right side of (12.5), the last term describes the torques of the base, which are zero because the base is not actuated, and the torques of the active joints  $\boldsymbol{\tau} \in \mathbb{R}^{12}$ .

Finally, the second to last term is the most relevant for leg odometry and its factors have variable dimensions depending on the robot configuration (humanoid or quadruped) and the number of legs in contact. Let  $c$  be the number of legs in contact and  $d$  be the number of active joints per a single leg ( $d = 3$  for quadrupeds,  $d = 6$  for humanoids). Then,  $\mathbf{J}_b \in \mathbb{R}^{dc \times 6}$  is the Jacobian matrix mapping the base twist to feet velocities, which depends on the forward kinematics of the robot and its absolute body orientation in the inertial frame  $\mathcal{F}^w$ .  $\mathbf{J}_q \in \mathbb{R}^{dc \times 12}$  is the stack of Jacobians described in (12.3) whose arrangement depends on the type of robot and number of contact legs. For example, a humanoid standing on both legs will have:

$$\mathbf{J}_q = \begin{bmatrix} \mathbf{J}_1(\mathbf{q}) \\ \mathbf{J}_2(\mathbf{q}) \end{bmatrix} \quad (12.6)$$

For quadrupeds, since the leg can pivot around the contact point, only the linear

velocity Jacobian  $\mathbf{J}_v$  is used for  $\mathbf{J}_q$ . For example, a quadruped standing on the first, third, and fourth leg, will have:

$$\mathbf{J}_q = \begin{bmatrix} \mathbf{J}_{1,v}(\mathbf{q}) \\ \mathbf{J}_{3,v}(\mathbf{q}) \\ \mathbf{J}_{4,v}(\mathbf{q}) \end{bmatrix} \quad (12.7)$$

The last term to consider is  $\mathbf{f} \in \mathbb{R}^{dc}$ , which represents the collection of all the forces and/or torques acting at each foot in contact with the ground. For a quadruped with all feet on the ground,  $\mathbf{f}$  is the stack of four linear forces:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \mathbf{f}_4 \end{bmatrix}. \quad (12.8)$$

For a humanoid with both feet on the ground, it contains the linear forces and torques acting on the three axes of the contact point:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \boldsymbol{\tau}_1 \\ \mathbf{f}_2 \\ \boldsymbol{\tau}_2 \end{bmatrix}. \quad (12.9)$$

As explained in more detail in Section 12.3.4, we can infer leg contact using the aforementioned forces and torques.

### 12.1.6 Joint Sensing

Similarly to fixed-base manipulators, the joints and the end effectors of legged robots are equipped with a variety of sensors, which are primarily used for planning and control [1009]. We briefly describe the most important sensors that are used also for leg odometry, namely encoders, force/torque sensors, and contact sensors.

#### 12.1.6.1 Rotary Encoders

Rotary encoders are electro-mechanical devices that convert an angular position of a rotating shaft into an analog or digital signal. In legged robots, they enable us to measure the joint angles and determine the robot kinematics, but they can also be found in other components. For example, mechanical LiDARs use them to measure the azimuthal angle of the beam array (see Chapter 8).

Encoders can be categorized depending on the principle of operation (optical or magnetic), the type of reading (absolute or incremental), and type of output (analog or digital). The most adopted type on legged robots are absolute and relative optical digital encoders; other encoders are discussed in more detail in related literature [716].

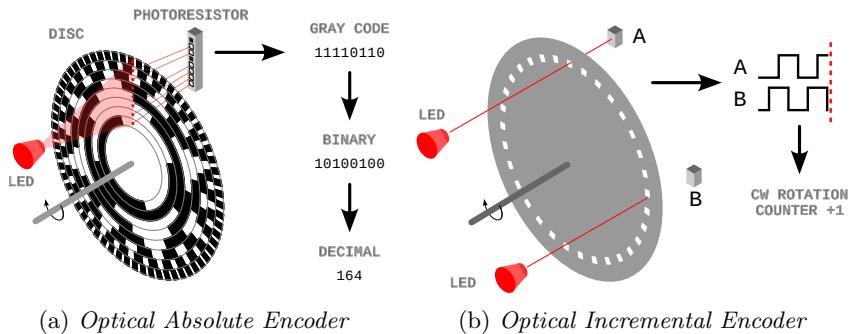


Figure 12.3 (a) Principle of operation of an 8-bit optical absolute encoder. An IR light beam hits a rotating disc that masks light according to a specific pattern that encodes the angle of rotation of the disc. An array of photoresistors convert the absence or presence of the light in each sector to a binary number encoded with a Gray code. The Gray code is then translated into a decimal number representing the absolute angle of rotation. The encoder in this example has a resolution of  $360/256 = 1.41$  degrees. (b) Principle of operation of an optical incremental encoder. The two photoresistors, A and B, are placed with a 90 degrees phase shift. A rising edge on A followed by a falling edge on B indicates a clockwise rotation. The change from  $AB = 11$  to  $AB = 10$  causes the increment of the counter.

Absolute optical digital encoders (Figure 12.3a) measure the joint angles in an absolute manner —for the same joint configuration they will provide the same sensor readings. Their operation principle is that an IR light source (*e.g.*, a Light Emitting Diode (LED)) hits an array of sensitive elements (*e.g.*, photoresistors) disposed radially on the static part of the device. In between the light source and the sensitive elements sits a disc that rotates with the shaft. The disc is divided in concentric sectors that can be either opaque or transparent. The sectors are arranged according to a pattern that encodes a specific angular range to a binary number. The binary number is ordered according to the Gray code, which maps consecutive natural numbers to binary numbers that always differ by only one bit, which reduces chances of reading errors. The process is illustrated in Figure 12.3a for an 8-bit encoder. The angular resolution of the device is determined by the number of bits (*i.e.*, the number of concentric sectors) used to make the binary word encoding the angle. For instance, for an 8-bit sensor there are 256 possible values, and then the angular resolution is  $360/256 = 1.41$  degrees.

#### *incremental encoders*

*Absolute optical encoders*

Incremental optical encoders (Figure 12.3b), conversely, measure relative angular changes with respect to the *initial configuration*, *i.e.*, they measure a zero angle when they are turned on, and then measure the angle relative to that reference point. The angle is calculated by adding or subtracting small angle increments, depending on the direction of rotation. Instead of relying on Gray codes, they operate by using a simpler codewheel made of an opaque material with regular slots placed radially, such that a single photoresistor A produces a square wave over time

when the disc rotates at constant speed. A second photoresistor  $B$  is placed at 90 degrees out of phase with the first one. The 2-bit word composing the two signals  $AB$  can have four different values at any given time, and the transition between them is used to determine the direction of rotation and whether the count has to be increased or decreased [716]. Because of their simpler construction and lower cost, high resolution incremental encoder have been used to compute the joint angle after a lower resolution absolute encoder measured the initial angle [984]. Even though they are still in use, incremental encoders are being rapidly replaced by absolute encoders, whose technology development improves their resolution while reducing their cost.

#### 12.1.6.2 Force and Torque Sensors

Force and torque sensors are devices that convert a linear force (applied to a point on a surface) or a mechanical torque (applied to a shaft) into an electrical signal. They are primarily used for torque control on the actuators or to sense the interaction between the end effector and the environment. In legged locomotion, each step involves forces being applied to the ground that need to be measured (directly or indirectly) so that stance legs can be identified.

The principle of operation for both type of quantities (force and torque) is typically the same, with different geometries: the internal surfaces of the sensors are shaped in a way that would slightly deform under stress along the direction where the force needs to be measured. Glued to those surfaces is a flexible variable resistive element, the *strain gauge*. The electrical resistance of the strain gauge changes proportionally to the amount of deformation it sustains, with compression (extension) causing a reduction (increase) in resistivity. Figure 12.4 shows an example with strain gauges applied to a load cell to measure linear force [716]. To measure torque, a series of strain gauges is applied to flexible spokes of a wheel connected to a motor shaft.

*strain gauge*

To measure all forces and torques acting on an end effector, 6-axis sensors are available, containing strain gauges in a number of configurations sufficient to measure forces and torques in all directions. These are commonly used for manipulation tasks, but can be also found on humanoids feet to directly measure the interaction with the ground.

With the introduction of cost effective dynamic legged robots, which was made possible by a backdriveable motor design [543], torques can be estimated from the motor currents, which are proportional to the torque by a constant factor.

#### 12.1.6.3 Contact Sensors

Since the main use for force and torque sensors in leg odometry is to determine the stance legs, alternative cost effective solutions are contact sensors, whose output is a binary number indicating whether a certain foot is in contact or not. This type

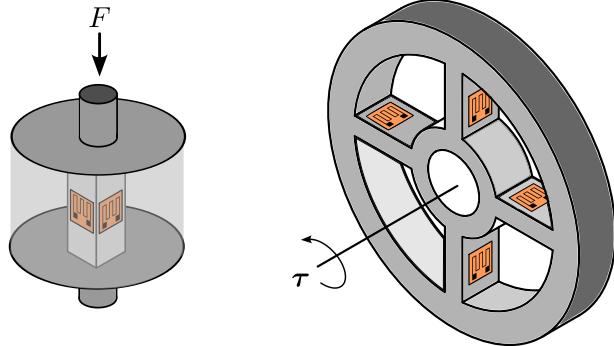


Figure 12.4 Principle of operation of force and torque sensors. A strain gauge is applied on compressible or flexible elements when under load. Inside the loadcell (on the left), a strain gauge is applied such that a compression would be detected as a reduction in resistivity. Inside the torque sensor (on the right), several strain gauges are applied at the flexible elements (the spokes of a wheel). When torque is applied, the elements would deform by flexion. The presence of multiple strain gauges allow to work out the magnitude and direction of the torque *e.g.*, by sensing a compression on one side of the spoke and an elongation on the other side.

of sensor was mainly developed for small to medium quadruped robots, whose feet consist of a spherical or circular rubber sole.

The main types of contact sensors are optical [397] or mechanical [791]. Optical contact sensors are conceptually similar to encoders: a LED-photodiode pair sense light through a small aperture. When the foot is in contact, the surface of the foot deforms enough to create a displacement of a masking panel that occludes the aperture, allowing the system to detect the contact. Mechanical contact sensors use a simple pushbutton switch hidden inside the sole that is pressed when enough force is exerted on the foot.

The main disadvantage of contact sensors is the relatively slow response time compared to costly force/torque sensors. In addition, they suffer from the same drawbacks of F/T sensors: they require to route cables up to the foot and they are at risk of damage due to the main impacts they have to sustain. For these reasons, they are mostly available only for small sized quadrupeds mostly designed for indoor operations.

## 12.2 Motion Estimation

Given the joint states and kinematics of the robot, we are now interested in computing the incremental motion of the robot's base. This can be mainly done in two ways: using the forward kinematics to obtain a relative pose between two time instants, or using the differential kinematics to estimate the robot's velocity instant-

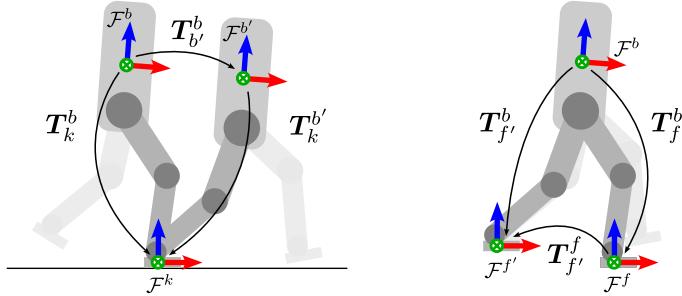


Figure 12.5 How leg odometry works with ideal contact. Left: Assuming the contact frame  $\mathcal{F}^k$  is rigidly attached to the ground (in yellow), we can determine the relative motion of the robot's body. Right: Alternatively, we can represent how the leg moves with respect to the body frame (yellow) in two consecutive instants.

taneously. In both cases, the underlying assumption is that a newly formed contact frame remains stationary for a certain amount of time.

### 12.2.1 Relative Pose Estimation

Figure 12.5 shows a simplified example of a humanoid robot walking along the  $zx$ -plane. The robot's base is represented at two consecutive time instants with the frames  $\mathcal{F}^b$  and  $\mathcal{F}^{b'}$ ; the foot frames and the joint positions at the same two times are defined similarly. Because the contact frame is stationary, when the foot frame and the contact frame coincide, the amount of displacement the robot's body experiences while moving forward is the same as the foot experiences moving backwards from the robot's body:

$$\mathbf{T}_{b'}^b = \mathbf{T}_k^b (\mathbf{T}_k^{b'})^{-1} = (\mathbf{T}_{f'}^f)^{-1} = (\mathbf{T}_f^b)^{-1} \mathbf{T}_f^b = \mathbf{fk}(\mathbf{q}')^{-1} \mathbf{fk}(\mathbf{q}) \quad (12.10)$$

Equation (12.10) creates a mapping between the joint states and the relative pose of the robot. While a concatenation of these relative poses would effectively provide a valid motion estimate by *dead reckoning* from joint sensing only [944], this is only possible during the stance phase. Further, to be applied on robots with point feet, this requires at least three feet in contact with the ground at all times, making it impractical for quadrupedal platforms.

To overcome these issues, the standard approach for quadrupeds [88] is to augment the state in (12.1) with the positions  $\mathbf{c}_i = \mathbf{t}_k^w \in \mathbb{R}^3$  of the contact frames expressed in world coordinates and associated to each leg of the robot. For humanoids, since they have ankles, the orientation of the contact points  $\mathbf{B}_i = \mathbf{R}_k^w \in \text{SO}(3)$  can also be added to the state [945].

Further, since there is no guarantee that at any given time there are a sufficient number of legs in contact (*e.g.*, a *gallop* gait has phases where all the legs are off

the ground), it is also commonly assumed that an IMU is present, so the angular velocity  $\boldsymbol{\omega}$  in (12.1) is disregarded, and the IMU biases are included as part of the state instead (see Chapter 11).

With all the previous considerations, the corresponding states of interest for quadrupeds and bipeds are then defined as:

$$\mathbf{x}_k = [\mathbf{t}_k \quad \mathbf{R}_k \quad \mathbf{v}_k \quad \mathbf{b}_k^a \quad \mathbf{b}_k^\omega \quad \mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{c}_3 \quad \mathbf{c}_4]^T \quad (12.11)$$

$$\mathbf{x}_k = [\mathbf{t}_k \quad \mathbf{R}_k \quad \mathbf{v}_k \quad \mathbf{b}_k^a \quad \mathbf{b}_k^\omega \quad \mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{B}_1 \quad \mathbf{B}_2]^T \quad (12.12)$$

where (12.11) represents the state of a quadruped robot, whilst (12.12) corresponds to a humanoid robot. This enables us to precisely express the motion estimate relationship from (12.10) for an arbitrary  $i$ -th leg:

$$\mathbf{T}_k^b = \mathbf{f}\mathbf{k}(\mathbf{q}) = (\mathbf{T})^{-1}\mathbf{C}_i \quad (12.13)$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (12.14)$$

is the pose of the base in the fixed frame, whereas

$$\mathbf{C}_i = \begin{bmatrix} \mathbf{B}_i & \mathbf{c}_i \\ \mathbf{0} & 1 \end{bmatrix} \quad (12.15)$$

is the pose of the foot contact in the fixed frame. Expanding (12.13) leads to:

$$(\mathbf{T})^{-1}\mathbf{C}_i = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{B}_i & \mathbf{c}_i \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^T\mathbf{B}_i & \mathbf{R}^T\mathbf{c}_i - \mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (12.16)$$

Rearranging (12.16), we can define the following components corresponding to the upper blocks of right-hand side matrix:

$$\mathbf{f}_p(\mathbf{q}) = \mathbf{R}^T(\mathbf{c}_i - \mathbf{t}) \quad (12.17)$$

$$\mathbf{f}_R(\mathbf{q}) = \mathbf{R}^T\mathbf{B}_i \quad (12.18)$$

where  $\mathbf{f}_p(\mathbf{q})$  denotes the relative position change of the foot in the fixed frame, and  $\mathbf{f}_R(\mathbf{q})$  the relative orientation change, as a function of the joint angles. Please note that for quadrupeds we only use (12.17), since we cannot obtain an orientation estimate from point feet.

Equations (12.17) and (12.18) are the basic leg odometry expressions used as measurements within estimation frameworks such as filtering or factor graphs. These will be further described in Section 12.4.

### 12.2.2 Velocity Estimation

The differential kinematics function from (12.3) can be used to get a direct velocity measurement from each stance leg. This approach is widely adopted on quadrupeds

[89, 143, 577] and less frequently on bipeds [1082]. The advantages are that velocity measurements can be easily (pre)integrated into a filter (or factor); they do not retain any history to avoid position error build up [315] and they do not need to keep track of extra states (contact poses or positions). However, we must point out that the joint velocities are usually numerically differentiated from joint positions, possibly degrading the estimation performance due to rounding errors.

Following a similar procedure as with the relative pose measurements, we aim to describe the velocity relationships that hold while a leg in rigid contact with the ground moves. First, we observe that the contact point  $k$  must be stationary for stance legs, hence the velocity of the contact point seen from the fixed frame must be zero:

$$\mathbf{v}_k^w = \mathbf{0}. \quad (12.19)$$

Furthermore, the velocity of the contact point  $k$  must coincide with the velocity of the foot  $f$ , also described by the Jacobian matrix (12.3):

$$\mathbf{v}_k^b = \mathbf{v}_f^b = \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}}. \quad (12.20)$$

Since the angular velocity of the robot is measured by the IMU, we focus on the linear velocity only. From (12.19) and (12.20), we can determine the robot's body velocity as:

$$\begin{aligned} \mathbf{v}_k^w &= \mathbf{v}_b^w + \boldsymbol{\omega}_b^w \times \mathbf{t}_k^b + \mathbf{v}_k^b \\ \mathbf{0} &= \mathbf{v}_b^w + \boldsymbol{\omega}_b^w \times \mathbf{f}_p(\mathbf{q}) + \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} \\ \mathbf{v}_b^w &= -\boldsymbol{\omega}_b^w \times \mathbf{f}_p(\mathbf{q}) - \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} \end{aligned} \quad (12.21)$$

where we take into account the additional linear velocity caused by the lever arm between the base and the foot [270]. (12.21) maps the absolute velocity of the robot to the forward and differential kinematics functions and can therefore be used as a measurement update or factor in a graph, as we will describe in Section 12.4. Note that since we conventionally express velocities in body coordinates, these can be obtained by using the robot orientation  $\mathbf{R} = \mathbf{R}_b^w$ .

Up to now we assumed that the stance legs are known. In the next section, we describe different methods to identify them.

### 12.3 Contact Estimation

The definition of contact estimation varies with the application. For example, in collaborative robotics, the end effector of a manipulator might be considered in contact as soon as it is “touching” something, *i.e.*, there is a non negligible external force exerted on it. For leg odometry, a foot can be considered in contact only when the contact point is stationary over time; on robots with point feet, this means ensuring it does not slip.

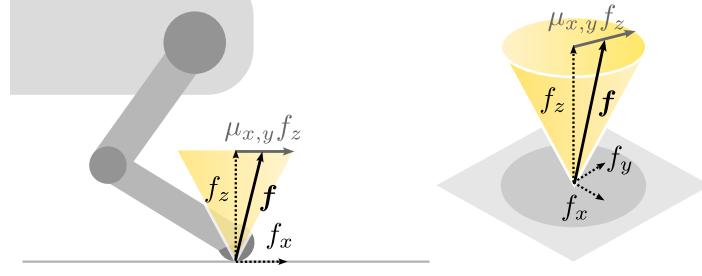


Figure 12.6 The contact point on a quadruped’s leg. A leg can be considered in stance when the force  $\mathbf{f} = [f_x, f_y, f_z]^\top$  applied by the foot stays within the friction cone.

Technically, a foot does not slip when the vertical component of the Ground reaction Force (GRF),  $f_z$ , is within the friction cone [946]:

$$\sqrt{f_x^2 + f_y^2} \leq \mu_{x,y} f_z \quad (12.22)$$

where  $f_x$  and  $f_y$  are the tangential components of the GRF with respect to the contact plane, which depend on the local morphology of the terrain.  $\mu_{x,y}$  is the friction coefficient, which depends on the mechanical properties of the ground and the foot touching it.

Humanoids with flat feet can also exert a torque on the ground. This introduces an additional condition to the non-slipping condition (12.22), which requires that the foot does not rotate:

$$\begin{bmatrix} -\tau_y/f_z \\ \tau_x/f_z \end{bmatrix} \leq \begin{bmatrix} CoP_x \\ CoP_y \end{bmatrix} \quad (12.23)$$

$$|\tau_z| \leq \mu_z f_z \quad (12.24)$$

where  $\tau$  is the contact torque,  $\mu_z$  is the rotational coefficient of friction, and  $CoP_x$ ,  $CoP_y$  denote upper limits of the components of the *center of pressure*, which define the contact support polygon bounds that are functions of contact surface geometry.

Since a sufficiently-high normal force  $f_z$  would guarantee that inequalities (12.22)-(12.24) are satisfied regardless of the other contact wrench (force and torque) dimensions, the most adopted approach for contact estimation is to simply threshold  $f_z$ . Then, the only differences from an implementation point of view are how the force is measured/estimated (*i.e.*, contact sensors, F/T sensors, joint sensing, IMUs), and the specific characteristics of the robot.

### 12.3.1 With Contact Sensors

Contact sensors implicitly threshold  $f_z$  in hardware, as they are tuned such that the binary signal they provided is only activated when the measured force exceeds

the nominal  $f_z$ . This is the simplest case, as the leg odometry can directly rely on the binary state provided by these sensors.

### 12.3.2 With Force/Torque Sensors

When F/T sensors are present on the foot,  $f_z$  can be measured directly over time. This permits to associate specific force patterns to events that are not just binary. For example, a small but rising force that lasts for more than a certain time is an indication that the foot is striking the ground but not yet in stance. Conversely, a force that falls below a certain value (*e.g.*, half of the expected load for one leg) means the foot is about to break the contact and it is therefore not reliable. In both cases, the information coming from that leg need to be discarded or its associated uncertainty increased [315].

### 12.3.3 With IMUs

As seen in Chapter 11, IMUs are inexpensive sensors that provide acceleration and rotational velocity measurements. While we typically use them to measure these quantities with respect to the robot's body, we can also use them on the legs or feet. Since any force applied to the foot (*e.g.*, during a touch down) would cause a change to its acceleration, some works used them to implicitly detect the stance legs [1227, 946, 726]. The main advantage of this approach is that the sensor is not sustaining an impact directly, so it is less likely to break, at the cost of additional signal processing to effectively detect such acceleration changes.

### 12.3.4 From Joint Torque Sensing

While it might seem straightforward to add additional sensors at the feet to detect contact, the different robot morphologies, design, and integration challenges might not make it always possible. In this case, the GRF can be estimated from the joint torques by exploiting the robots' dynamics (see (12.1.5)).

Using a quadruped platform as an example, we can exploit the block-wise structure of  $\mathbf{J}_q$  to compute the force at the end effector from (12.5) as:

$$\mathbf{f}_i = -(\bar{\mathbf{J}}_{i,v}^\top)^{-1}(\boldsymbol{\tau}_i - \mathbf{h}_{q,i} - \mathbf{F}^\top \dot{\mathbf{v}}) \quad (12.25)$$

where:  $\mathbf{f}_i \in \mathbb{R}^3$  and  $\boldsymbol{\tau}_i \in \mathbb{R}^3$  are the GRF and the torque leg  $i$ ;  $\bar{\mathbf{J}}_{i,v}$  is the non-zero block  $i$ -th foot Jacobian  $\mathbf{J}_v$  (which for quadrupeds is a square matrix);  $\mathbf{F} \in \mathbb{R}^{3 \times 3}$  is one of the blocks of the mass matrix;  $\mathbf{h}_{i,q} \in \mathbb{R}^3$  is the vector of centrifugal/Coriolis/gravity torques for leg  $i$ .

Note that the estimate for  $\mathbf{f}_i$  can only be in base coordinates. However, to recover the actual GRF there are two pieces of information missing:

- the *local inclination of the terrain*, which the orientation of the contact force depends on. While the ankle joints of a humanoid can give a good approximation, for quadrupeds the orientation of the contact frame cannot be determined without exteroceptive sensing, but can be inferred heuristically from the other feet in contact (*e.g.*, by fitting a plane through them); [329].
- the *friction coefficient*, which the horizontal components of the force depend on. The friction coefficient depends on the material the robot is stepping on, so it can only be known a priori or inferred from the amount of slippage the robot is experiencing [513].

In general, establishing the contact states from joint sensing remains an open problem, and several techniques have been developed to detect contact in a probabilistic fashion, (*e.g.*, by combining also kinematics as well as dynamics of the robot [497]) or by using learning methods (see Section 12.6).

## 12.4 Using Leg Odometry for State Estimation

Now that we have specified the main steps required to obtain leg odometry measurements, in this section we describe how to integrate them into an estimation framework to solve the state estimation problem. The predominant estimation solutions are based on filtering approaches, which combine IMU and leg odometry at the high frequency required for closed-loop control. The factor-graph-based smoothing approaches described in the previous chapters have only been adopted on legged platforms in the last few years. However, their focus has not been on providing estimates for control but rather lower frequency estimates for mapping, which benefit from *slower* sensors such as LiDARs or cameras.

Regardless of the method, for optimally fusing leg odometry with other sensor modalities we need to quantify its associated uncertainties. This is the first topic we will cover before presenting the filtering and smoothing approaches.

### 12.4.1 Encoder Noise Propagation

The main sources of uncertainty in leg odometry are the robot's joints encoders, which measure the joint positions and are affected by noise. This noise can be modeled as an additive zero-mean Gaussian term  $\boldsymbol{\eta}_q \in \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_q)$ , such that the true value  $\mathbf{q}$  and the measured value  $\tilde{\mathbf{q}}$  are related as follows:

$$\tilde{\mathbf{q}} = \mathbf{q} + \boldsymbol{\eta}_q. \quad (12.26)$$

Since the forward and differential kinematics functions involve rotations, they are therefore nonlinear and they will not preserve the Gaussian properties of the encoder noise. However, as done in previous chapters, we can consider a first-order

approximation—which is locally linear and preserves Gaussianity—using the Jacobian function [439]:

$$\mathbf{f}_p(\mathbf{q} + \boldsymbol{\eta}_q) \approx \mathbf{f}_p(\mathbf{q}) + \mathbf{J}_c(\mathbf{q})\boldsymbol{\eta}_q \quad (12.27)$$

where  $\mathbf{J}_c(\mathbf{q})$  is the *body manipulator Jacobian*, *i.e.*, the same as the manipulator Jacobian  $\mathbf{J}(\mathbf{q})$  but expressed in the contact frame.

The same Gaussian assumption can also be applied to velocity measurements affected by encoder noise  $\boldsymbol{\eta}_q$  and encoder velocity noise  $\boldsymbol{\eta}_{\dot{q}}$  [1188], considering that:

$$\mathbf{J}(\mathbf{q} + \boldsymbol{\eta}_q)(\dot{\mathbf{q}} + \boldsymbol{\eta}_{\dot{q}}) \approx \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} + \frac{\partial}{\partial \mathbf{q}} (\mathbf{J}(\mathbf{q})\dot{\mathbf{q}}) \boldsymbol{\eta}_q + \mathbf{J}(\mathbf{q})\boldsymbol{\eta}_{\dot{q}}. \quad (12.28)$$

From (12.27) and (12.28), the extra terms multiplied by the noise terms can simply be grouped into a single term, since they are all linear combinations of a Gaussian term for a given encoder measurement.

### 12.4.2 Factor Graph Smoothing

To generate locomotion behaviors while avoiding falls and other catastrophic failures, legged robots have strict real-time control and high-frequency state estimation requirements. Historically, those requirements were met by using nonlinear variants of the Kalman Filter, such as the Extended Kalman Filter (EKF) [88, 143], the Unscented Kalman Filter (UKF) [89], or the Invariant-EKF [441, 1252, 667]. These types of filter would typically fuse together high-frequency sensor data, such as inertial and kinematics, to feed the controller. Exteroceptive sensor updates within the control loop have also been demonstrated [144] but those are normally relegated to mapping and planning purposes.

One limitation of Kalman filtering-based methods is that they are designed to have a process model in addition to the measurement model. When such a model is not available, it is usually replaced by a constant velocity model or, more often, IMU propagation. This suggests that factor-graph-based methods are a more general approach, as they consider both process models and measurement models in a general manner—as a relationship between states and measurements.

Factor-graph-based methods for legged systems mainly differ in the number of estimated states and the time horizon. When only two consecutive states are considered, a factor graph resembles a filter; the Two-State Implicit Filter (TSIF) [92] is an instance of this case. When the window is increased, instead of estimating only the most current state as the TSIF, the factor graph has the ability to correct a history of past states within a time window. The frequency of the states considered is a design decision: adding more frequent states at a high frequency (*e.g.*, IMU rate) simplifies the design of the estimator but it requires to reduce the time window to keep the computational requirements bounded. Conversely, longer time horizons

with a fixed number of states can be achieved by preintegrating measurements, as showed for IMU measurements in Chapter 11.

We next present two examples from the related literature that illustrate how preintegration theory and the leg odometry concepts previously introduced are leveraged in a factor graph estimation framework. We particularly focus on the case of contact preintegration for bipeds [440], and velocity bias preintegration for quadrupeds [1188]. In both cases, the measurements from Section 12.2 will be reformulated in terms of residuals and covariances for the factors of the graph.

#### 12.4.2.1 Contact Preintegration

Contact preintegration aims to integrate the relative motion increments from the kinematics of a humanoid robot, and add them as factors that link two humanoid states, defined as in (12.12). This idea was presented by Hartley *et al.* [440], and the proposed factor graph is shown in Figure 12.7a.

The factors are generally standard: a prior factor (in black) anchors the graph, while a preintegrated IMU factor (orange) introduces the motion prior from the IMU. Additionally, for humanoids we add a forward kinematics factor (in green) that constrains the pose of the contact frames, while the contact preintegration factor (in blue) encodes the relative motion between contact states of the two legs.

*forward kinematics factor*

**Forward Kinematics Factor.** The forward kinematics factor relates the pose of the contact frame at the feet to the pose of the robot, both expressed in the inertial frame, via the forward kinematics of the stance leg.

By plugging the encoder noise from (12.27) into the relative pose measurement of (12.17) and (12.18), we can define the following residual and covariance for the forward kinematics factor:

$$\mathbf{r}_{\mathcal{F}} = \text{Log}(\mathbf{C}_i^{-1} \mathbf{T} \mathbf{fk}(\tilde{\mathbf{q}})) \quad (12.29)$$

$$\boldsymbol{\Sigma}_{\mathcal{F}} = \mathbf{J}_c(\tilde{\mathbf{q}}) \boldsymbol{\Sigma}_q \mathbf{J}_c^T(\tilde{\mathbf{q}}) \quad (12.30)$$

where the residual enforces that the difference between the contact frame and the robot frame are close to the forward kinematics, given the uncertainty propagated from the encoders' noise.

*contact preintegration factor*

**Contact Preintegration Factor.** The contact preintegration factor adds an additional constraint on the contact point. Ideally, if there is no slip on the stance leg and the pose of the contact frame should remain unaltered; in practice, slip occurs and can be modeled as Gaussian noise added to the velocities of the contact point. The contact preintegration factor models how the contact point can change between two time instants due to this noise.

Technically, given two consecutive states  $\mathbf{x}_i$  and  $\mathbf{x}_j$  at times  $t_i$  and  $t_j$ , respectively, the following relationship holds:

$$\Delta \tilde{\mathbf{B}}_{ij} = \mathbf{B}_i^T \mathbf{B}_j \text{Exp}(\delta \boldsymbol{\theta}_{ij}) = \mathbf{I} \quad (12.31)$$

$$\Delta \tilde{\mathbf{c}}_{ij} = \mathbf{B}_i^T (\mathbf{c}_j - \mathbf{c}_i) + \delta \mathbf{d}_{ij} = \mathbf{0} \quad (12.32)$$

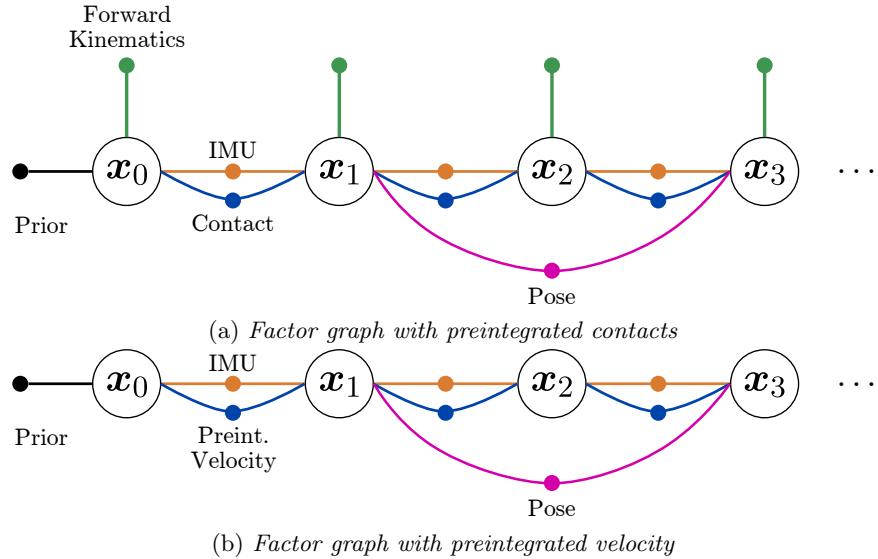


Figure 12.7 Factor graph formulations for preintegrated contact (top) and velocity (bottom). Additional measurements (*e.g.*, from exteroceptive sensors, which constrain two states) can be easily added as additional factors (magenta).

where we have used the rotational and translational components of the contact frames  $\mathbf{C}_i$  and  $\mathbf{C}_j$  as stated in (12.15). The terms  $\delta\theta_{ij}$  and  $\delta\mathbf{d}_{ij}$  are *preintegrated contact noise* terms, which are introduced to model the uncertainty on the contact point velocity as a zero-mean Gaussian variable [440]. The preintegrated contact factor is then given by the following rotation and translation residuals and covariance:

$$\mathbf{r}_c = \begin{bmatrix} \text{Log}(\mathbf{B}_i^\top \mathbf{B}_j) \\ \mathbf{B}_i^\top (\mathbf{c}_j - \mathbf{c}_i) \end{bmatrix} \quad (12.33)$$

$$\boldsymbol{\Sigma}_c = \begin{bmatrix} \boldsymbol{\Sigma}_w & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_v \end{bmatrix} \Delta t_{ij} \quad (12.34)$$

where we have rearranged and stacked (12.31) and (12.32) into a single vector residual. The covariance  $\boldsymbol{\Sigma}_c$  is made of the time integration of the contact angular covariance  $\boldsymbol{\Sigma}_w$  and linear velocity covariance  $\boldsymbol{\Sigma}_v$  over  $\Delta t_{ij}$ .

As a last note, an important limitation of the contact preintegration factor is that it is only valid for the same stance leg, during the stance phase. Henceforth, it is not valid for the switching dynamics of a legged platform. This has been addressed in follow up work [439], by modeling and properly handling the contact frame switches, enabling preintegration among different legs.

### 12.4.2.2 Velocity Preintegration

As mentioned previously, the kinematics of point feet platforms—such as quadruped robots—cannot constrain the relative 6 DoFs between two states. This impedes the use of the forward kinematic and contact preintegration factors recently introduced. Alternatively, we can exploit the linear velocity measurements from leg odometry, reviewed in Section 12.2.2, and preintegrate them to obtain additional factors that constraint the relative change of the robot’s pose [1188, 577].

*velocity factor* *preintegration* **Velocity Preintegration Factor.** This factor assumes that the instant linear velocity of the body can be determined from leg odometry using (12.28) and it is affected by Gaussian noise terms  $\boldsymbol{\eta}_v$  and  $\boldsymbol{\eta}_\omega$ :

$$\tilde{\mathbf{v}} = -\mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} - \boldsymbol{\omega} \times \mathbf{f}_p(\mathbf{q}) + \boldsymbol{\eta}_v. \quad (12.35)$$

Assuming the robot has constant body linear velocity between times  $t_i$  and  $t_j$ , we can preintegrate the velocity measurements to obtain:

$$\Delta\tilde{\mathbf{t}}_{ij} = \Delta\mathbf{t}_{ij} + \delta\mathbf{p}_{ij} = \sum_{k=i}^{j-1} [\Delta\tilde{\mathbf{R}}_{ik}\tilde{\mathbf{v}}_k \Delta t] + \delta\mathbf{p}_{ij}, \quad (12.36)$$

where, similarly to the preintegrated contact factors,  $\delta\mathbf{p}_{ij}$  is a *preintegrated velocity noise term* [1187, 577]. Then, the preintegrated velocity factor and associated covariance are given by:

$$\mathbf{r}_v = \mathbf{R}_i^\top (\mathbf{t}_j - \mathbf{t}_i) - \Delta\mathbf{t}_{ij} \quad (12.37)$$

$$\boldsymbol{\Sigma}_{v,ij} = \sum_{j=1}^{k=i} \boldsymbol{\Sigma}_{v,ik} + \mathbf{A}\boldsymbol{\Sigma}_v\mathbf{A}^\top, \quad (12.38)$$

with the matrix  $\mathbf{A} = \Delta\tilde{\mathbf{R}}_{ik}\Delta t$ .

### 12.4.2.3 Handling of Multiple Measurements

In the previous sections we have considered only one measurement per leg, without considering what to do when multiple legs are in contact at the same time. The presence of multiple legs in contact potentially provides redundancy and robustness, but increases the risk of inconsistencies (*e.g.*, when different legs provide conflicting information). The simplest approach adopted by some works (*e.g.*, [315]) is to pick only the leg that is deemed to be most reliable, discarding the information from the others.

Another intuitive approach is to treat each leg (and their measurements) independently. This is easier when the contact poses (or positions) are explicitly part of the state [440, 577]. When this is not the case, the velocity measurements simultaneously acquired by all legs in stance can still be treated as independent, but it is often preferable to average them into one single measurement to reduce the computational load on the filter or factor graph [1188].

### 12.4.3 Integration with Exteroceptive Sensors for SLAM

As we have seen in the previous sections, leg odometry provides an additional way to compute incremental motion between two consecutive states. Their main use is to improve the odometry estimate, such that the SLAM system building on top of it (*e.g.*, a ) can benefit from low drift edges between nodes, which translate into less abrupt corrections during loop closures.

The integration of additional sensors, such as LiDAR and cameras, is naturally handled by both filtering and smoothing approaches by simply adding more measurements to the former and factors to the latter. There are however subtle details to be considered while doing so. Fusing measurements from multiple independent sources, each one operating at different frequencies, levels of noise, and failure rates, is not trivial.

If a sensor modality breaks the zero-mean Gaussian noise assumption, or fails completely, the status of the filter (or factor graph) can be compromised. For this reason, also motivated by the DARPA SubT challenge, there has been a surge in loosely coupled methods that run different subsystems in parallel (*e.g.*, Visual-Inertial, Legged-Inertial, and LiDAR-Inertial) while triaging their outputs and select the best estimate from each subsystem [299, 557].

The alternative to loosely coupled methods are tightly coupled ones. In [1188] a fixed-lag smoother was used to fuse leg odometry with IMU, cameras and LiDAR in the same factor graph. In this case, to overcome the problems related to inconsistencies between the different types of factors or sensor failures, the triaging happens directly into the factors: if a sensor modality fails, the factor is simply not added to the graph. In addition, to handle noise that is not zero-mean Gaussian, robust cost functions can be used within factors.

## 12.5 Open Challenges

In previous sections we have covered how to generally compute leg odometry and fuse it with other sensor modalities. We made a number of assumptions that are often not valid in practice, and relaxing some of them remains an open problem. We briefly introduce some of the resulting open problems below.

### 12.5.1 Leg Deformation

The leg odometry equations we have seen throughout the chapter all assumed that the robot was a perfectly rigid body. When this assumption is not valid, leg odometry measurements will be biased, because the forward kinematics function computes the ideal position of the end effector and not the real one (see Figure 12.8). Similarly, when the contact point does not move, but forces are applied to it such that the legs bend, the joint angles change. When the problem occurs for short peri-

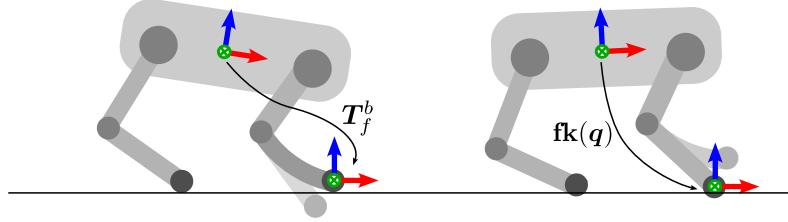


Figure 12.8 Example of leg deformation on a quadruped. The real transformation between the robot base and the contact point is shown at the left. Since the forward kinematics assumes the robot’s legs are rigid, it incorrectly estimates an upward motion, as shown on the right.

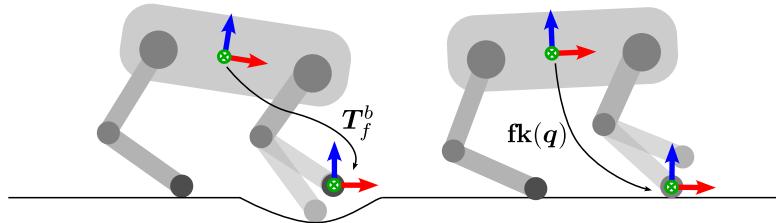


Figure 12.9 Example of ground deformation with a quadruped. The robot initially touches the ground which is flat. While keeping the contact state on, the ground deforms and the foot sinks into it (left). As the joint angles change while the foot goes down, this motion is interpreted as an upward motion from the initial touchdown point (right)

ods of time, detecting the impact by analyzing the force profile and rejecting the measurements during those periods is a strategy adopted in the past [143].

Since bipeds tend to have longer legs, the problem of leg flexibility can be even worse on such platforms. One way to approach it would be to exploit the correlation between the leg load and the flexibility (intuitively, the more a leg is loaded, the more it will flex) by carefully modelling the bending properties of the robot considering its structural geometry and properties. This approach is however complex and typically does not generalize well.

Instead, the most adopted approach is to integrate additional IMU sensors located on the links and estimate the link orientation compared to the joint readings [1125].

### 12.5.2 Non-rigid Contacts and Slippage

If the robot is walking on soft or collapsible ground, the leg stretches penetrating the terrain (Figure 12.9, left). When the contact is first detected before the ground starts its deformation and since the contact point is assumed to be stationary, this is interpreted by the forward kinematics as an upward motion (Figure 12.9, right). Even if the effect is similar to leg deformation, in this case it is the assumption of

zero velocity of the contact point to be broken [314], since the contact point moves downwards as the ground deforms. This problem is similar to slippage, when a foot is considered in contact with the ground but the forces it exerts on the terrain violate (12.22) and/or (12.24).

In this case, an exteroceptive sensor such as a camera is needed to make the velocity of the contact point observable in non-degenerate motions and robot configurations [1082]. The velocity of the contact can be explicitly tracked as an additional state in [1188], or as the derivative of the feet positions [577].

## 12.6 Further Readings & Recent Trends

In this last section, we focus on some of the recent trends in state estimation and legged robotics. While some of them address part of the open challenges discussed in the previous section, such as contact estimation, others also represent significant paradigm shifts in the current techniques —particularly those aided by learning algorithms.

**Learning-based Contact Estimation.** We discussed how contact estimation assumes rigid contact which is easily violated by situations such as slippage, soft terrain, or leg deformation. Given the challenges of accurately modeling these problems, it has been proposed to use data-driven methods for contact estimation. These approaches generally aim to learn a binary signal that determines when contact was established. This has been demonstrated in a supervised manner by learning contact classifiers [143], but also in an unsupervised fashion via clustering [945], where proprioceptive sensing (joint and inertial) provide the main signals for the models.

With the rise of deep learning methods, it has been proposed to use neural network architectures to determine the contact state of the feet. This has shown better generalization to a wider set of structured and unstructured environments [667]. Vision-based haptic sensors, which capitalize on the progress of machine learning and computer vision [639], are another direction that shows promise to improve force and contact estimates [1000].

**End-to-End Learning.** While high-frequency leg odometry and proprioceptive state estimation were developed to achieve closed-loop model-based locomotion control, the current progress in reinforcement learning (RL) has challenged their necessity. RL-based locomotion controllers showed that only the body velocity and orientation are required for locomotion, which can be provided explicitly by a standard proprioceptive state estimators [498], or even raw data from joint and inertial sensing [636, 760].

While the latter might question the need for state estimation, this is explained by the manner these particular RL-based controllers are trained. The training objective aims to track velocity commands, emulating the way in which the robot will be controlled by a human operator or planning system. To achieve this, the locomotion controller only needs to know the orientation of the base with respect to the gravity

vector, as well as the instantaneous body velocity. As seen in Chapter 11, these quantities are fully observable from inertial data, hence can be implicitly estimated during training.

In contrast, another current trend in locomotion learning aims to achieve advanced mobility skills to navigate the world, by learning locomotion controllers that are able to traverse different obstacles and reach goals relative to a starting position —*robot parkour* being an example [1308, 464]. Achieving this navigation behavior does require access to an odometry estimate, since the robot needs to keep track of the progress towards the goal in an inertial frame. This suggests that state estimation is still needed to achieve more complex locomotion and navigation tasks.

A few works have proposed to learn state estimation as part of the locomotion policy learning process [515], and explicitly estimate variables such as the body velocity, feet height, and contact state. While this has only been used for locomotion purposes, it can be a promising alternative to obtain more accurate leg odometry estimates for proprioceptive state estimation or odometry factors in SLAM.

**Humanoid Robots.** Humanoid robots embed part of the dreams that have motivated the development of robotics —creating artificial agents able to do the *dull, dangerous, and dirty* tasks that humans prefer not to do. Having a *human-like body* should —in principle— enable them to seamlessly work in human-oriented environments, using tools, devices, and even vehicles designed for people’s use.

The DARPA Robotics Challenge, briefly introduced in Section 12.1.1, has been one of the main efforts in this direction. The diverse set of tasks, involving locomotion on rough terrain but also tool handling and driving vehicles, presented several challenges towards this goal. However, after it ended in 2015, most of the efforts in legged robotics focused on quadrupedal platforms instead —which presented clear advantages in control and robustness, motivating their adoption for industrial inspection and monitoring. It was not until 2021 when a commercial interest in humanoid platforms arose again, motivated by the optimism and fast-pacing progress in artificial intelligence (AI), as well as the success of quadrupedal robots.

Diverse companies have recently aimed to develop new humanoid platforms as a way to *embody* AI systems in the real world. Humanoid robots have been targeted to solve complex tasks in delivery, warehousing, and manufacturing —working side-by-side with people, in highly demanding environments. This presents different challenges that push the topics covered in this chapter in directions currently unexplored. Problems such as long-term, accurate and reliable whole-body estimation need to be solved for humanoid robots to be able to achieve tasks in last-mile delivery problems. Intermittent contacts, from the feet but also the trunk, arms, and hands are expected when handling parcels or other objects in a warehouse setting. Similarly, compliance is required when operating close to people in order to be safe —this also requires relaxing the rigid contact assumptions we made in this chapter.

**Acknowledgments**

The authors thank Michele Focchi (University of Trento) for his advice in preparing parts of this chapter.



## **PART THREE**

### FROM SLAM TO SPATIAL AI



# III

## Prelude

Marc Pollefeys Ayoung Kim, Frank Dellaert,  
Timothy Barfoot, Luca Carlone, and Daniel Cremers

In Part I, we explored the foundational principles of SLAM, followed by an in-depth look at SLAM systems across various sensor modalities in Part II. Building on this foundation, Part III extends SLAM into the broader domain of Spatial AI. We begin this part with a perennial question that has persistently challenged the SLAM community.

### *Is SLAM a solved problem?*

The answer could be yes —if we limit ourselves to small, static, indoor environments rich in features. Yet such controlled settings are far from where SLAM’s true promise lies. We imagine SLAM as the guiding light for robots venturing into realms beyond human reach —dark, hostile, and uncharted. We dream of underwater explorers diving to depths no human could ever fathom, mapping the mysteries of the abyss. We want SLAM to navigate robotic explorers on Mars, analyzing alien terrains and environments. We yearn for SLAM to lead rescue missions in disaster-stricken zones, locating survivors and saving lives. And, of course, we want SLAM to be deployed in highly dynamic, human-centric, and city-scale environments, say in a self-driving car in dense down-town traffic.

Parts I and II covered the fundamental concepts necessary to reach this final goal, which is intertwined with many real-world challenges. One thing is certain: we need greater intelligence in perceiving and understanding the world, making the transition to spatial AI essential.

### III.1 Spatial Artificial Intelligence

What is the key difference between conventional SLAM and spatial AI? The first thing to note is the integration of recent advances in deep learning for spatial understanding. Transitioning to learning-based frameworks enhances the ability to handle complex, difficult-to-model scenarios, resulting in more adaptive and robust performance. Beyond performance improvements, spatial AI will transform and expand space representation by enabling novel view synthesis, enriching maps with semantic information, and securing flexibility from dynamic scene changes.

The emerging role of foundation models in computer vision and robotics must be harnessed for advancing spatial AI. At the same time, it is crucial not to overlook the importance of real-world deployment, particularly by addressing the computational aspects of the system components.

### **III.2 Spatial AI Applications**

Spatial AI promises to enable devices to understand the environments they navigate in, to assist people or enable (physical) autonomous agents to carry out tasks. While multi-modal foundation models trained on images and accompanying text can provide a general explanation of observed scenes, many tasks require to build up richer persistent scene representations. In fact, many image foundation models struggle altogether with spatial understanding in images as they only provide a holistic scene description. By combining SLAM and semantic image understanding capabilities, persistent 3D semantic scene representations can be built and kept up to date, creating spatio-temporal memories of environments. Spatial AI, particularly when combined with SLAM, is transforming various domains by enabling devices to understand and interact with their physical environment in real time. Even more can be achieved when building a shared spatial representation through crowd mapping. This section describes the application of Spatial AI to AR/VR glasses, humanoid robots, self-driving vehicles, drones, industrial automation, emergency response and healthcare.

#### ***III.2.1 AR/VR Glasses***

*AR/VR*

In augmented and virtual reality, Spatial AI with SLAM enables immersive, context-aware experiences by allowing AR/VR glasses to understand and interact with the physical world in real-time. SLAM helps these devices map their surroundings and track their position within that space, which is essential for anchoring virtual objects accurately in the user's environment. For example, in AR applications, this allows digital content to remain fixed to real-world surfaces as the user moves, enhancing realism and usability. In VR, it supports inside-out tracking, eliminating the need for external sensors and enabling more natural movement and interaction.

Moreover, Spatial AI can interpret semantic information from the environment—like identifying walls, furniture, or people—which allows for more intelligent and adaptive applications. This is particularly useful in collaborative AR scenarios, where multiple users interact with the same virtual content in a shared space, or in accessibility tools that provide spatial cues to visually impaired users.

### III.2.2 AI Glasses

AI glasses represent a specialized evolution of AR wearables, integrating Spatial AI to deliver real-time contextual intelligence directly to the user. Unlike general-purpose AR/VR headsets, AI glasses are designed for lightweight, continuous use in everyday environments. With SLAM, these glasses can localize themselves and map the surrounding space, enabling persistent spatial awareness and interaction with digital content anchored to the real world.

These capabilities unlock a range of applications —from real-time translation and navigation assistance to object recognition and task guidance. For instance, in industrial settings, AI glasses can provide step-by-step instructions for machinery, while in retail, they can provide product information or inventory data as the user looks around. When paired with voice interfaces and edge AI processing, these glasses become powerful tools for hands-free computing, enhancing productivity, accessibility, and situational awareness in both professional and consumer contexts.

### III.2.3 Humanoid Robots

For humanoid robots, Spatial AI is foundational to achieving autonomous, *humanoid* like navigation and interaction. These robots must perceive and understand complex, dynamic environments —such as homes, offices, or hospitals— and SLAM enables them to build and update maps of these spaces while simultaneously localizing themselves within them. This is critical for tasks like fetching objects, guiding people, or performing inspections.

Beyond navigation, Spatial AI allows humanoid robots to reason about their surroundings. For instance, they can recognize and avoid obstacles, understand spatial relationships (*e.g.*, “the cup is on the table”), and plan actions accordingly. When combined with vision and language models, this spatial understanding enables more intuitive human-robot interaction, where robots can follow natural language commands like “go to the kitchen and bring me the red mug.”

### III.2.4 Self-Driving Vehicles

In autonomous vehicles, Spatial AI and SLAM are key to safe and efficient navigation in both structured and unstructured environments. SLAM allows vehicles to create high-fidelity maps of their surroundings in real time, which is especially valuable in areas where GPS is unreliable or unavailable, such as tunnels or dense urban canyons. These maps help the vehicle localize itself precisely and detect changes in the environment, such as construction zones or temporary obstacles.

Spatial AI enhances this by adding semantic understanding —identifying lanes, traffic signs, pedestrians, and other vehicles. This enables more sophisticated decision-making, such as predicting pedestrian intent or negotiating complex intersections.

In combination with other sensors like LiDAR and radar, SLAM-based Spatial AI forms the backbone of perception systems that allow self-driving cars to operate safely and autonomously in diverse conditions.

### ***III.2.5 Drones and Aerial Robotics***

*drones*

In aerial robotics, Spatial AI with SLAM enables drones to autonomously navigate complex environments without relying on GPS. This is especially valuable in indoor, underground, or densely forested areas where GPS signals are weak or unavailable. SLAM allows drones to build 3D maps of their surroundings in real time, avoid obstacles, and plan efficient flight paths. Combined with semantic understanding, drones can identify objects of interest —such as power lines, crops, or structural damage— making them invaluable for inspection, agriculture, and disaster response.

### ***III.2.6 Industrial Automation and Warehousing***

*industrial automation*

In smart factories and warehouses, Spatial AI empowers mobile robots and automated guided vehicles (AGVs) to operate safely and efficiently alongside human workers. SLAM enables these systems to localize themselves and dynamically update their maps as the environment changes —such as when inventory is moved or new obstacles appear. Spatial AI also supports task-level reasoning, allowing robots to understand spatial relationships (*e.g.*, shelf locations, aisle widths) and optimize logistics operations like picking, packing, and delivery.

### ***III.2.7 Emergency Response***

*search and rescue*

In emergency response scenarios, Spatial AI and SLAM are invaluable for search and rescue operations, disaster assessment, and recovery efforts. Drones and ground robots equipped with SLAM can navigate hazardous environments, such as collapsed buildings or flood zones, to locate survivors and assess damage. These systems can create real-time maps of the affected areas, providing critical information to first responders and enabling more efficient and safer operations, in particular if those have access to AR/VR glasses or other wearables.

Spatial AI also supports the deployment of autonomous vehicles and robots to deliver supplies, medical aid, and communication equipment to areas that are difficult to access. By understanding and mapping the environment, these systems can navigate complex terrains and avoid obstacles, ensuring timely and accurate delivery of essential resources.

### ***III.2.8 Healthcare and Assistive Technologies***

*healthcare*

Spatial AI is increasingly used in assistive devices for people with disabilities. For

example, wearable devices equipped with SLAM can help visually impaired users navigate unfamiliar environments by providing real-time spatial cues and obstacle warnings. Semantic understanding of the environment is critical to provide effective assistance. In hospitals, autonomous service robots use Spatial AI to deliver medications, transport supplies, and guide patients, reducing staff workload and improving operational efficiency.

### **III.3 How to Read Part III?**

The organization of this part follows a topic-wise chapter categorization of how SLAM is advancing toward Spatial AI, leveraging recent learning-based approaches.

In Chapter 13, we introduce how deep learning techniques can enhance and extend conventional SLAM systems. Recent progress in novel view synthesis and its integration into SLAM frameworks is discussed in Chapter 14. Approaches for handling dynamic environments and structural changes are presented in Chapter 15. In Chapter 16, we explore the transition from purely metric maps to semantically enriched representations for higher-level scene understanding. The integration of large language models to support spatial reasoning is examined in Chapter 17. Finally, we investigate the computational structures and system-level design considerations essential for enabling Spatial AI in Chapter 18.

# 13

## Boosting SLAM with Deep Learning

Zachary Teed, Jia Deng, Boris Chidlovskii, Jérôme Revaud, Felix Wimbauer and Daniel Cremers

*SLAM and Deep Learning*

Following the introduction of large, curated datasets [606], researchers have deployed deep neural networks for a multitude of challenges ranging from image segmentation [687] and optical flow estimation [278] to protein prediction [388, 527]. These systems are able to leverage large, labeled datasets and use highly over-parameterized neural networks with powerful internal representations to learn a mapping from the set of inputs to the space of labels. A natural question is how we can extend the success of deep learning to SLAM.

As we saw in the previous chapters, modern SLAM systems contain many interconnected components including feature detection, feature matching, outlier rejection, and relocalization. One promising line of work has investigated replacing individual components with learned systems. For example, networks for feature detection [267], descriptors [214], matching [1049, 1134, 968, 1135, 668] and outlier rejection [968] can be trained on large datasets and can often outperform their classical counterparts. Once these components are trained, they can be plugged back into the full SLAM pipeline to produce more accurate and robust systems. Deep neural networks can also be trained to predict useful 3D properties directly from input images such as single image depth [91, 1225, 302] or relative pose between pairs of frames [1226]. Depth or relative pose can then be naturally integrated into existing SLAM systems, *e.g.*, as additional loss in a factor graph [1226].

One issue with the modular approach is that errors from individual components can feed into each other and compound. Increasingly, deep networks have been shown capable of combining more and more of the pipeline into end-to-end trainable modules, including the integration of optimization problems such as BA as neural network layers [1079, 1065, 1152] (also see Chapter 4). The DUS3R [1161] line of works takes this idea even further by replacing the full stack with a network that predicts pointmaps (*i.e.*, a parametrization of a structured 3D point cloud) directly from uncalibrated images.

In this chapter, we will discuss several promising ways to bring the power of neural networks into the SLAM pipeline. An overview of some of the works presented in this chapter is shown in Figure 13.1. In many ways these are techniques that try to combine the best of both worlds. Some feed depth, pose, and uncertainty

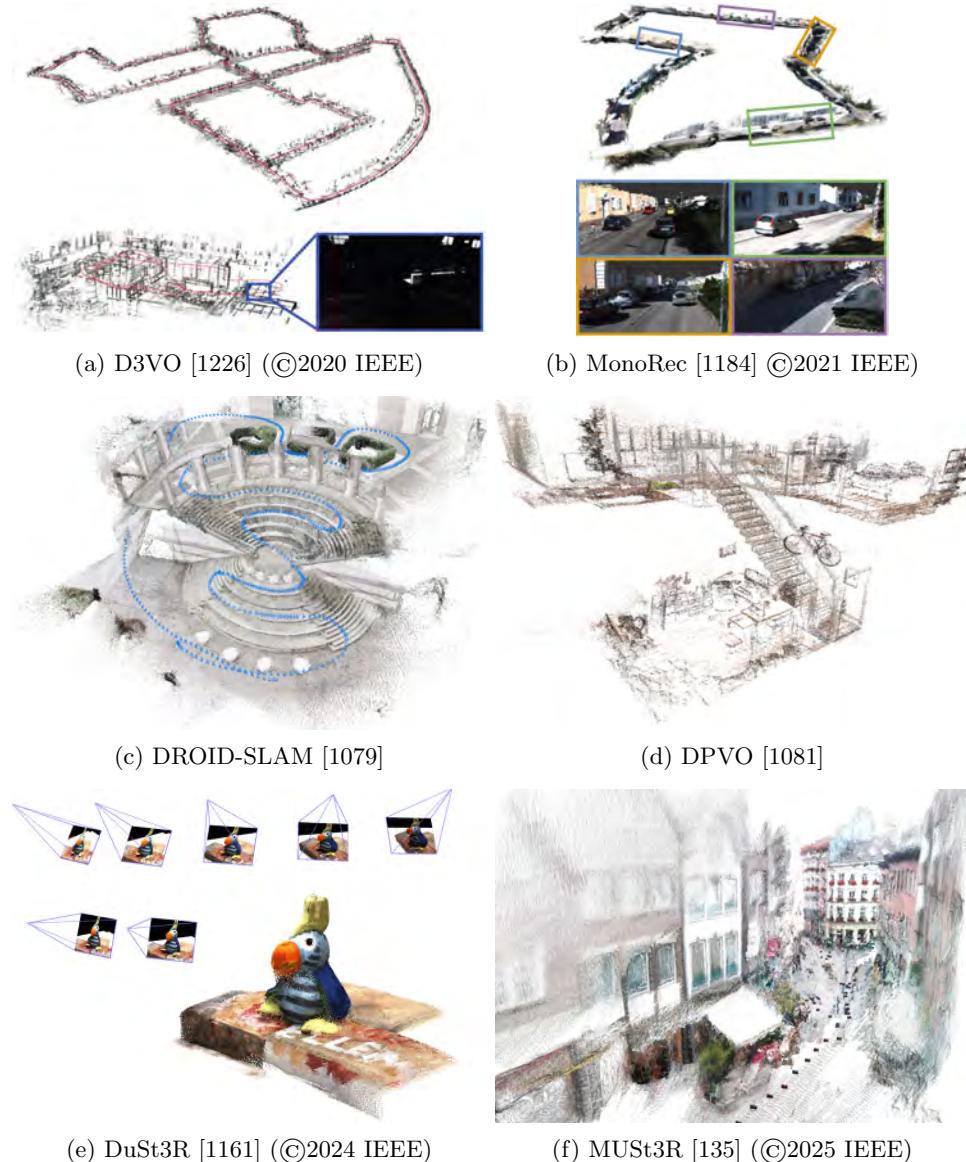


Figure 13.1 Over the last years, a multitude of efforts have been made to bring the predictive power of deep networks into visual SLAM. In this chapter, we will review a number of these approaches.

predictions from neural networks into an optimization-based SLAM pipeline. Some combine end-to-end learned components with factor graphs. And some explore novel ways of estimating inter-frame motion and 3D structure with suitably trained net-

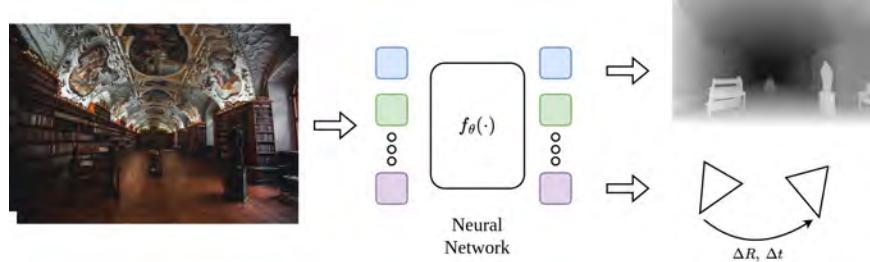


Figure 13.2 A neural network can be trained to predict 3D properties, such as depth or relative pose, from an input image or collection of images. These 3D properties can be used to augment SLAM systems.

works. The resulting methods can significantly boost visual SLAM methods, providing higher precision and robustness and often generalize well beyond their training domain. More in detail, Section 13.1 reviews how to use deep learning for depth and camera pose prediction. Section 13.2 discusses applications to feature matching and tracking. Section 13.3 focuses on the use of differentiable optimization to train SLAM end-to-end, and describes DROID-SLAM, which can be considered the first successful system showing this integration. The following chapters discuss transformer-based architectures that go from uncalibrated images to pointmaps, including DuSt3R (Section 13.4), MAST3R (Section 13.5), and their integration into SLAM and SfM pipelines (Section 13.6). The chapter closes with a review of open challenges (Section 13.7).

### 13.1 Deep Learning for Depth and Camera Pose

Deep neural networks can be trained to learn a mapping from an input space  $\mathcal{Z}$  (*e.g.*, images) to the label space  $\mathcal{X}$  (*e.g.*, depth). Modern neural networks consist of a large number of differentiable layers which can be composed into end-to-end trainable systems. The parameters of the network are optimized using variations of gradient descent over a loss function which measures how well the prediction of the network matches the ground-truth label.

With modern depth sensors and laser scanners, it is now possible to collect large amounts of 3D data. For example, depth sensors or LiDARs can be used to collect images paired with depth maps. In addition, rendering engines can be used to produce large-scale, synthetic 3D datasets. As shown in Figure 13.2, neural networks can be trained on this data to predict 3D properties directly from input images, such as depth or relative camera pose. In this section, we show how the predictions from these networks can be used to augment SLAM systems.

### 13.1.1 Deep Learning for Depth Prediction

Given a large collection of image and depth pairs, neural networks can be trained to predict depth (or some parameterization of depth) directly from a single input image. Eigen et al. [302] first showed that convolutional neural networks could be used to predict depth from a single image. Over the years, single-image depth networks have continued to advance on account of both improved network architectures and the expanding collection of 3D datasets. Modern depth networks often use a pre-trained vision transformer (ViT) backbone [280] with a convolutional decoder to up-sample the coarse resolution ViT features and produce a final depth estimate [1159, 480]. To improve generalizability, these networks are often trained on combinations of over 20 datasets including both synthetic and real data.

*depth prediction*

Depth predictions have many immediate applications in SLAM, particularly in the monocular case. Monocular SLAM systems typically only produce reconstructions up to a scale factor and are subject to scale drift. Yang et al. [1225] and Hu et al. [480] showed that metric depth predictions could be used to combat scale drift of a monocular SLAM system. Depth predictions can also be used to robustify systems and avoid failure cases in challenging situations such as pure camera rotation, changing camera parameters, and dynamic objects [658].

Furthermore, visual SLAM systems often only track a sparse subset of feature points, leading to sparse reconstructions of the scene. For example, areas with little texture are barely present in the resulting point cloud. While sparse point clouds are already useful, many downstream applications such as robot navigation or autonomous driving require a *dense* reconstruction. In the following, we provide examples of methods aiming to obtain denser 3D reconstructions.

**MonoRec: Monocular Dense Reconstruction [1184].** One way to densify the output of a sparse SLAM system is to predict dense depth maps for every frame of the video. A dense reconstruction, such as the one shown in Figure 13.1b, can be obtained by lifting all the pixels into 3D and accumulating them in a global reference frame using the poses produced by a SLAM system. However, simply using the depth maps produced by a depth network is suboptimal for a couple of reasons. First, single image depth networks tend to produce geometrically inconsistent outputs between adjacent frames. Second, many scenes contain dynamic objects which would introduce trailing artifacts in a static reconstruction. *MonoRec* [1184] improves geometric consistency by augmenting the depth network with a cost volume and filters dynamic objects by predicting a moving object mask.

*MonoRec*

To improve geometric consistency, for each frame  $I_t$ , *MonoRec* [1184] builds a cost volume  $C_t(d)$  aggregated over consecutive frames  $\{I_{t_1}, \dots, I_{t_n}\}$ . The cost volume is built by computing the photometric consistency of each pixel at a number of discrete depth steps  $d$  aggregated into a 3D volume. *MonoRec* uses a depth network which takes in both the input image  $I_t$  and its cost volume  $C_t(d)$  to predict depth. The cost

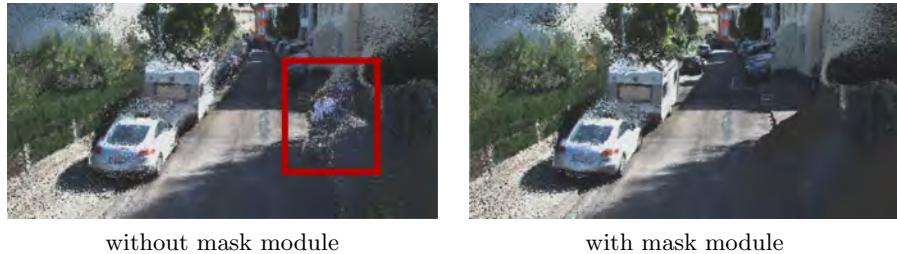


Figure 13.3 MonoRec: The mask module in the network architecture allows to filter out moving objects based on brightness inconsistency across time [1184] (©2021 IEEE).

volume provides the network with additional context for producing geometrically consistent depths.

Cost volumes assume a static world and provide wrong signals if an object in the scene is moving. Therefore, MonoRec relies on an additional masking network which predicts a moving object segmentation mask  $M_t$  from the cost volume. The moving object segmentation mask is used to filter out dynamic points when lifting the predicting depths into 3D. Figure 13.1b shows that the masking module can filter out moving objects in the dense reconstruction.

MonoRec is trained using the sparse 3D point predictions of the SLAM system. Since the *Mask* module can benefit from having accurate depth and the *Depth* module can benefit from accurate dynamic masks, joint training of the two components can further boost performance, yielding accurate and consistent depth maps.

#### *Behind The Scenes*

**Behind the Scenes (BTS)** [1185]. Dense point clouds, as produced by MonoRec [1184], can accurately represent the 3D world observed in the video. However, regions that were not occluded are consequently not part of the reconstructed 3D scene. *Behind the Scenes (BTS)* tackles this shortcoming by predicting a volumetric reconstruction instead of a depth map. Here, everything in the camera frustum is reconstructed, even the areas that are occluded in the current frame. In BTS, scenes are represented as volumetric density fields  $\phi(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ , which map any point  $\mathbf{x}$  in the camera frustum to a volumetric density  $\sigma$ . Through the established volume rendering technique, it is possible to synthesize any novel view from this scene representation. Overall, a volumetric reconstruction contains useful additional information for many downstream applications, especially in the autonomous driving and robotics domain.

#### 13.1.2 Deep Learning for Camera Pose Prediction

##### *camera pose prediction*

Similar to how a neural network can be trained to predict depth, a neural network can also be trained to predict camera pose between pairs of frames or video sequences. Letting  $I_1$  and  $I_2$  be a pair of input images, we can design a neural

network  $f_\theta$  which takes the two images as input and predicts the relative camera pose from  $I_1$  to  $I_2$  denoted  $f_\theta(I_1, I_2) = \mathbf{T}_1^2 \in \text{SE}(3)$ . Given ground-truth poses  $\mathbf{T}_1^{2*} \in \text{SE}(3)$ , we can define a loss

$$\mathcal{L}_{\text{pose}} = \|(\mathbf{T}_1^2)^{-1} \cdot \mathbf{T}_1^{2*}\|, \quad \text{where } \mathbf{T}_1^2 = f_\theta(I_1, I_2) \quad (13.1)$$

which measures the relative transformation between the predicted pose and the ground-truth pose. It is also possible to define other losses such as minimizing the rotation angle or computing distance between predicted and ground-truth quaternion representations of relative rotations.

Early works for camera pose estimation used networks which concatenated the two frames as input and regressed camera pose parameters such as axis-angle [1111] or Euler angles [1296]. Given the relative pose between pairs of frames, one can recover the full trajectory of the camera by chaining the poses together by composition. TartanVO [1164] improved relative posed estimation by augmenting the input images with predicted optical flow and camera intrinsics, leading to better generalization on downstream datasets.

One issue with two view pose prediction is that relative pose errors can compound for longer video sequences. DeepVO [1160] proposed using a recurrent neural network which takes in a sequence of frames as input and predicts a sequence of camera poses. DeepVO extracts features from the input video sequence to generate a sequence of feature maps. These feature maps are then processed by an LSTM to predict a sequence of camera pose estimates.

While these works showed promising results, directly predicting camera pose is often less accurate than more classical SFM or SLAM techniques where geometry is more deeply embedded in the system [1079]. Furthermore, these techniques have a tendency to overfit to their training distribution and require training on large combinations of datasets to encourage better generalization.

### 13.1.3 Unsupervised Learning of Depth and Camera Pose

Supervised learning of depth and pose require ground-truth 3D data which might not always be available or can be difficult to obtain. It turns out there are ways to train *pose* and *depth* networks without explicit 3D supervision. One common technique uses view synthesis as a supervision signal [387, 1296]. Given two images,  $I_1$  and  $I_2$  with relative poses  $\mathbf{T}_1^2$ , we can generate a new image from the viewpoint of  $I_1$  using pixels sampled from  $I_2$ .

To see how this is done, let  $\pi_c : \mathbb{R}^3 \mapsto \mathbb{R}^2$  be the camera projection which takes a 3D point to its 2D pixel coordinates and  $\pi_c^{-1} : \mathbb{R}^2 \times \mathbb{R}^+ \mapsto \mathbb{R}^3$  be the inverse projection function which takes a 2D pixel with depth  $d$  to its 3D point. Given these functions and relative pose  $\mathbf{T}_1^2$ , we can map a  $(\mathbf{x}^1, d^1)$  pair in  $I_1$  to its location in  $I_2$

$$\mathbf{x}^2 = \pi_c(\mathbf{T}_1^2 \pi_c^{-1}(\mathbf{x}^1, d^1)) = \omega_{1 \rightarrow 2}(\mathbf{x}^1, d^1) \quad (13.2)$$



Figure 13.4 DVS0: By feeding monocular depth predictions from a deep neural network into a classical SLAM method like DSO [308], the monocular method “Deep Virtual Stereo Odometry” (DVS0) [1225] achieved highly accurate camera trajectories that were on par with state-of-the-art stereo-based methods. The deep network was trained in a self-supervised manner using stereo videos and lead to significant boost in precision and the capacity to recover scaled reconstruction from a single moving camera.

barring occlusions and using  $\omega_{1 \rightarrow 2}$  as a shorthand. Given a depth map  $\mathbf{d}$ , (13.2) can be used to warp  $I_2$  into the viewpoint of  $I_1$  by defining the warped image

$$\tilde{I}_2(x) = I_1(\omega_{1 \rightarrow 2}(\mathbf{x}, \mathbf{d}(x))) \quad (13.3)$$

where  $I_1(\cdot)$  denotes bilinear interpolation. Since bilinear interpolation is locally differentiable with respect to pixel coordinates, it is possible to compute the derivative of the synthesized image  $\tilde{I}_2$  with respect to depth and camera pose. Godard et al. [387] showed that a view synthesis loss could be used to train a depth network on stereo images. Following works showed that this idea can be taken even further and jointly trained a *depth* and a *pose* network using view synthesis supervision [1296, 1126].

DVS0

**Deep Virtual Stereo Odometry (DVS0) [1225].** One limitation of monocular SLAM systems is that they tend to be less robust and more prone to drift than stereo systems. DVS0 showed that a classical visual odometry system, namely DSO [308], could be augmented using depth predictions from a neural network trained using view-synthesis supervision between stereo pairs of frames [387]. DVS0 works by taking the depth predictions from the neural network and adding an additional factor that encourages the consistency between the predicted depth maps and the depth computed by DSO as shown in Figure 13.4.

Using the depth predicted by the neural network improved the reconstruction quality along with the accuracy and robustness of the poses predicted by the VO system. Additionally, since the depth predictions were in metric units, the method was able recover scale-accurate reconstructions even from a single camera. This work was coined “virtual stereo odometry” since in the monocular application the deep network essentially hallucinated the effect of a second camera.

D3VO

**D3VO: Deep Depth, Deep Pose, and Deep Uncertainty for Monocular Visual Odometry [1226].** Beyond predictions of monocular depth, one can go further and also feed predictions of relative camera motion into the classical SLAM pipeline in order to ensure consistency of the recovered trajectory with the predic-

tions of camera motion for consecutive frames. Direct visual SLAM methods like DSO [308] rely on color consistency across frames to infer localization and mapping. In the real world, this color consistency is not always guaranteed—in particular for non-Lambertian surfaces like shiny, metallic, or glass surfaces, corresponding points will likely not have the same color. While it is conceivable to explicitly model the reflection properties of the world, this is computationally demanding and likely infeasible in a real-time setting.

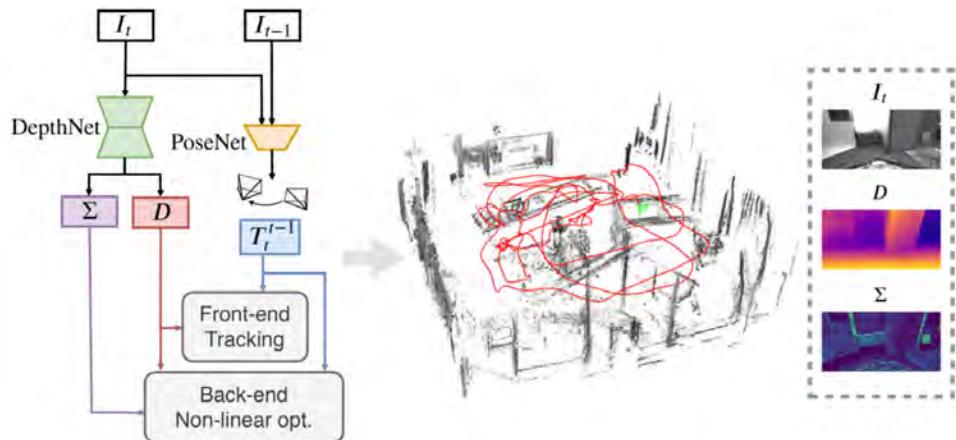


Figure 13.5 D3VO [1226] combines self-supervised neural networks for pose, depth, and uncertainty prediction with a direct SLAM system [308]. The pose and depth predictions from the networks are integrated into the SLAM system as additional factors in the factor graph, encouraging the state variables to the SLAM system to align with the predictions of the neural networks. By combining deep learning with factor graphs, D3VO benefits from the robustness of neural networks with the accuracy of direct SLAM. Image from [1226] (©2020 IEEE).

It turns out that one can train a neural network to predict locations where color is likely preserved across frames. Such predictions can also be fed into a direct SLAM pipeline, essentially down-weighting the residuals where color consistency is not expected. In this manner, D3VO [1226] feeds deep network predictions of monocular depth, relative inter-frame camera motion and uncertainty into a SLAM pipeline. As shown in Figure 13.5, D3VO uses a *depth* and a *pose* network trained using view synthesis losses in a self-supervised manner. The predictions enter on the frontend in form of respective factor graph terms, but also in the backend in the form of respective loss terms. By combining depth and pose predictions from a neural network with direct visual odometry [308], D3VO is able to achieve highly accurate camera trajectories and is robust to failure cases common for monocular VO.

### 13.2 Deep Learning for Feature Matching and Optical Flow

The previous sections showed how neural networks can be used to predict 3D information such as depth and camera pose directly from input images. This feed-forward approach contrasts with the typical SLAM pipeline, where inference typically takes the form of an optimization problem: solving for  $\mathbf{x}^{\text{MAP}}$  given the noisy measurements  $\mathbf{z}$  such as feature matches.

Predicting camera pose or depth is not an easy problem. To do so, the network essentially needs to learn multi-view geometry from scratch, such as the relationships between depth, camera pose, and camera intrinsics. An alternative to this approach is to predict the measurements  $\mathbf{z}$ , such as feature matches, instead of the state variables directly.

#### 13.2.1 Learning Feature Detectors and Descriptors

Indirect SLAM approaches rely on feature detectors and descriptors to extract a sparse set of visual measurements from the input images. First, salient regions of the images are identified, commonly referred to as *keypoints*. Then the local neighborhood of each keypoint is used to compute a vector descriptor which can be used to match keypoints between images.

Neural networks can be trained to replace the hand-designed feature detectors and descriptors commonly used in SLAM and SfM pipelines. For instance, SuperPoint [267] proposed using a convolutional neural network to both (1) detect salient image regions and (2) assign feature descriptors to the detected keypoints. Similarly, Universal Correspondence Network [214] showed that feature descriptors produced by a neural network can improve downstream feature matching accuracy. Such networks are typically trained using a *contrastive* loss, which encourages the feature descriptors for matching points to be similar and encourages the distance between the feature descriptors for non-matching keypoints to be large.

#### 13.2.2 Feature Matching

Given a set of keypoints extracted from a pair of images, neural networks can also be used to improve the matching of keypoints between images. SuperGlue [968] introduced the idea of using a neural network to guide feature matching. SuperGlue operated over keypoints extracted from a pair of input images. SuperGlue iteratively updated feature vectors associated with each keypoint with self and cross attention layers as shown in Figure 13.6. The network outputs an assignment matrix, providing the likelihood of each keypoint in the first image with each keypoint in the second image (with an additional column for unmatched keypoints). SuperGlue showed that the predicted keypoint matches could improve accuracy on several

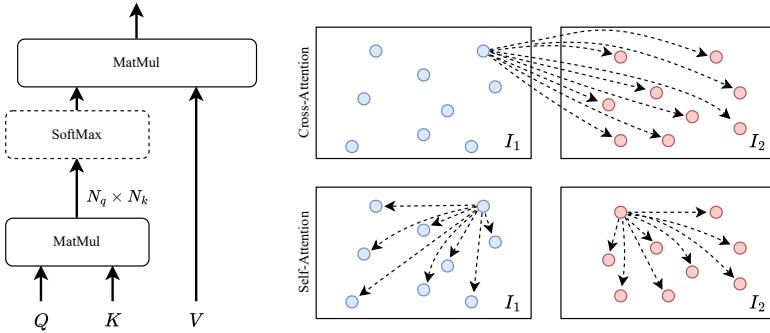


Figure 13.6 Illustration of Multi-Head self and cross attention commonly used in feature matching networks such as SuperGlue [968], LightGlue [668] and LoFTR [1049]. SuperGlue and LightGlue perform self and cross attention over feature vectors associated with image keypoints. LoFTR does not rely on keypoint detection and performs self and cross attention over dense feature grids.

downstream tasks such as pose estimation. LightGlue [668] improved the accuracy and speed of SuperGlue with a collection of architecture and training improvements.

One potential limitation of SuperGlue and LightGlue is that they operate over a sparse set of image keypoints. While this has the advantage of reducing computation, it cannot use the full image which can limit the number of high quality matches on difficult examples such as images with limited texture. LoFTR [1049] proposed a network for dense pixel matching. Similar to SuperGlue and LightGlue, LoFTR leveraged self attention and cross attention (Figure 13.6). In order to reduce compute, LoFTR performed dense matching over coarse resolution feature grids, then refined the predictions at higher resolution.

The feature matches produced by these systems can be directly integrated into indirect SLAM systems in the form of visual measurements and their corresponding factors. Deep feature matching can improve the robustness of SLAM systems in cases where traditional feature descriptors are not reliable.

### 13.2.3 Optical Flow and RAFT

Optical flow is closely related to the problem of feature matching. It is the task of estimating per-pixel motion between a pair of frames. This per-pixel motion—or flow field—can be used as measurements in a factor-graph-based visual SLAM system.

Originally, optical flow was formulated as an optimization problem with a cost function that balanced two terms: a *data* term and a *regularization* term. The data term encourages the alignment of visually similar image regions while the

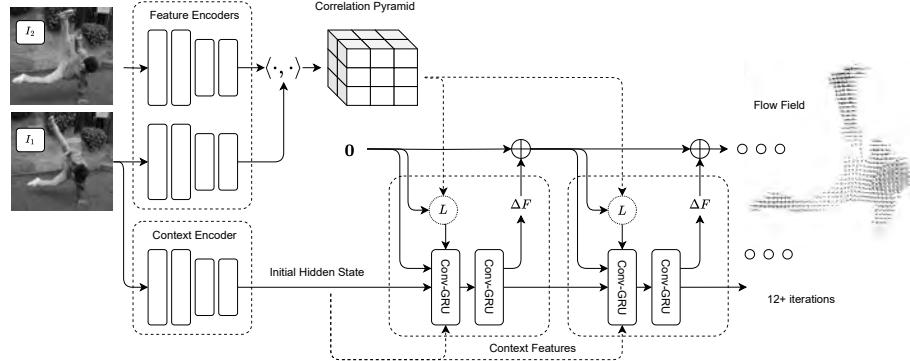


Figure 13.7 RAFT: Recurrent All Pairs Field Transforms for Optical Flow architecture. Features are extracted from the input images using a pair of residual networks. RAFT then builds a correlation volume by taking an all-pairs inner product between the features from  $I_1$  and the features from  $I_2$ . Starting from a flow field initialized at 0, RAFT iteratively uses the current estimate of optical flow to perform lookups from the correlation pyramid and applies a recurrent convolutional GRU to update the flow estimate.

RAFT

regularization term encourages physically plausible motion fields. In more recent years, deep learning has emerged as a promising alternative [278, 1048, 502, 904]. In this section, we describe one such approach, RAFT [1078] (Figure 13.7), as it serves as the basis for DROID-SLAM [1079] (Section 13.3). More recent optical flow networks have improved upon RAFT with the addition of multi-head attention and transformer blocks [494] such as those illustrated in Figure 13.6.

Figure 13.7 illustrates the overall design of RAFT. RAFT combines elements of both deep learning and traditional first-order optimization. Features are first extracted from the input images then used to build a 4D correlation volume. Starting with a flow field initialized at zero, the flow field is used to perform lookups from the correlation pyramid. The correlation features, alongside the current estimate of the flow field, are plugged into a Convolutional-GRU which produces an update to the flow field in addition to an updated *hidden state*. RAFT is trained on a combination of synthetic datasets with ground-truth optical flow [740, 278] but shows good generalization to real data [754].

**Feature Extractor.** The feature extractor is used to encode the input images into dense feature maps at lower resolution. RAFT uses a pair of feature extractors: (1) a appearance encoder  $g_\theta$  and (2) a context encoder  $c_\theta$  which map the input image with dimensions  $(h_{in}, w_{in}, 3)$  to a lower resolution feature map with shape  $(h, w, c)$  where  $c$  is the number of feature channels. By default, RAFT extracts features as 1/8 resolution so  $(h, w) = (h_{in}/8, w_{in}/8)$  but some other networks use higher resolution.

**Correlation Pyramid.** The appearance features are used to build a multi-

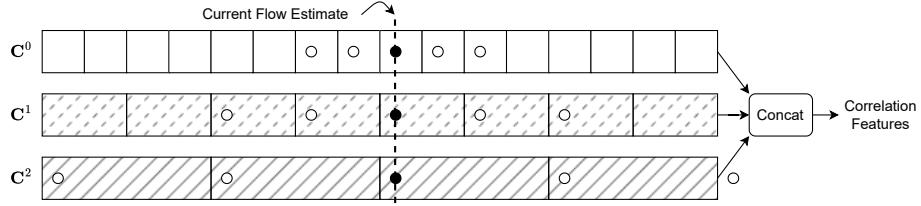


Figure 13.8 RAFT correlation lookup operator. This figure illustrates a 1D slice of the lookup operation. For the current flow estimate, RAFT constructs a local neighborhood and retrieves features from each level of the correlation pyramid which are then flattened and concatenated.

resolution correlation pyramid. The correlation volume is 4-dimensional tensor which has shape  $h \times w \times h \times w$  where  $h$  and  $w$  are the dimensions of the appearance feature map as defined above. The correlation volume  $\mathbf{C}$  is constructed by taking the inner product between all pixels in  $g_\theta(I_1)$  with all pixels in  $g_\theta(I_2)$

$$\mathbf{C}_{ijkl}(I_1, I_2) = \langle g_\theta(I_1)_{ij}, g_\theta(I_2)_{kl} \rangle = \sum_m g_\theta(I_1)_{ijm} \cdot g_\theta(I_2)_{klm}. \quad (13.4)$$

This operation can be implemented as a single matrix multiplication (by flattening the spatial dimensions and taking the transpose of the first argument) which is highly optimized on the GPU. RAFT doesn't just use a single correlation volume but constructs a multi-resolution correlation pyramid. The pyramid is constructed by iteratively performing average pooling on the last two dimensions of the correlation volume to produce a 4 level pyramid  $\{\mathbf{C}^0, \mathbf{C}^1, \mathbf{C}^2, \mathbf{C}^3\}$  where the  $l^{\text{th}}$  level volume  $\mathbf{C}^l$  has shape  $h \times w \times h/2^l \times w/2^l$ .

**Correlation Lookup.** RAFT defines a *lookup* operator which uses the current estimate of the flow  $\mathbf{f}$  to index from the correlation pyramid (Figure 13.8). For a given pixel  $(u, v)$  in  $g_\theta(I_1)$ , the flow field  $\mathbf{f}$  is used to compute its estimated correspondence in  $I_2$  by following the flow vector  $(u', v') = \mathbf{f}(u, v) + (u, v)$ .

RAFT then constructs a  $(2r + 1) \times (2r + 1)$  neighborhood grid around  $(u', v')$

$$N(u', v') = \{(u' + \Delta u, v' + \Delta v) \mid \Delta u, \Delta v \in \{-r, \dots, r\}\} \quad (13.5)$$

where  $r$  is a hyper-parameter which defines the lookup radius. The neighborhood grid is used to index from  $\mathbf{C}^0$  via bilinear interpolation. Features are sampled from the lower resolutions  $\mathbf{C}^l$  by using neighborhood grids centered around the rescaled correspondence coordinates  $N(u'/2^l, v'/2^l)$ . The final feature vector is formed by concatenating the features from each level as shown in Figure 13.8 and contains both fine resolution features about small displacements and coarse resolution information about large displacements.

**Update Operator.** The RAFT weight-tied update operator estimates of flow fields  $(\mathbf{f}^1, \dots, \mathbf{f}^N)$  starting from the zero field. With each iteration of the update

operator, it produces an update direction which is applied to the current estimate to produce the next estimate  $\mathbf{f}^{k+1} = \delta \mathbf{f}^k + \mathbf{f}^k$ .

The update operator uses gated activation unit based on the GRU [217] where the linear layers are replaced with convolutions. The convolutions allow information to be spatially propagated over the image and weight tying and bounded activations encourage convergence to a fixed point. Each iteration takes the following four inputs (each in the form of a 2D feature map): (1) the *hidden state* from the previous iteration, (2) the *context features*, (3) the current *flow estimate* and (4) *correlation features* from the lookup operator. These features are taken as input to the Conv-GRU which produces an update hidden state and the flow update  $\delta \mathbf{f}^k$ .

#### 13.2.4 Optical Flow as Visual Measurements

In a static scene the optical flow between a pair of frames is a function of depth and camera pose. To see this, let  $I_i$  and  $I_j$  be two images with known camera poses  $\mathbf{T}_i^w$  and  $\mathbf{T}_j^w$  (as camera to world transformations). Given a pixel  $(u, v)$  with depth  $d$ , we can compute the induced optical flow by first unprojecting the point  $(u, v)$  to depth  $d$ , then transforming the 3D point from frame  $i$  to  $j$ , followed by projecting onto camera  $j$

$$\hat{\mathbf{f}}(u, v) = \boldsymbol{\pi}(\mathbf{T}_i^j \cdot \boldsymbol{\pi}^{-1}(u, v, d)) - (u, v) \quad \mathbf{T}_i^j = (\mathbf{T}_j^w)^{-1} \cdot \mathbf{T}_i^w. \quad (13.6)$$

where  $\boldsymbol{\pi} : \mathbb{R}^3 \mapsto \mathbb{R}^2$  which takes a 3d point in pixel coordinates and projects it to pixel coordinates, and the inverse projection function  $\boldsymbol{\pi}^{-1} : \mathbb{R} \times \mathbb{R}^2 \mapsto \mathbb{R}^3$  which back-projects pixel coordinates  $(u, v)$  with depth  $d$  to its 3D coordinate.

Here we assume pinhole cameras but complicated camera models with radial or tangential distortion can also be used if they are undistorted as a preprocessing step which is possible assuming the distortion model is known. For notational convenience, we overload  $\boldsymbol{\pi}$  and  $\boldsymbol{\pi}^{-1}$  with vectorized versions of these operators that act on dense  $h \times w$  grids of pixels  $(\mathbf{u}, \mathbf{v})$ , such that the flow field can be represented in the form

$$\mathbf{h}_{ij}(\mathbf{T}_i^w, \mathbf{T}_j^w, \mathbf{d}_i) = \boldsymbol{\pi}(\mathbf{T}_i^j \cdot \boldsymbol{\pi}^{-1}(\mathbf{u}, \mathbf{v}, \mathbf{d}_i)) - \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \quad (13.7)$$

where  $\mathbf{d}$  is taken to be a  $h \times w$  dense depth map.

#### 13.2.5 Estimating Pose and Depth from Optical Flow

We can use factor graphs to go in the reverse direction: using estimates of optical flow to solve for depth and camera pose. Consider a set of images  $\{I_1, \dots, I_n\}$  and let  $\mathcal{G}$  be a graph with edges connecting images with flow estimates (Figure 13.9a), such that  $(i, j) \in \mathcal{G}$  means that we have an estimate of optical flow between images  $I_i$  and  $I_j$ . Let's denote the flow estimate between frames  $I_i$  and  $I_j$  as the measurement

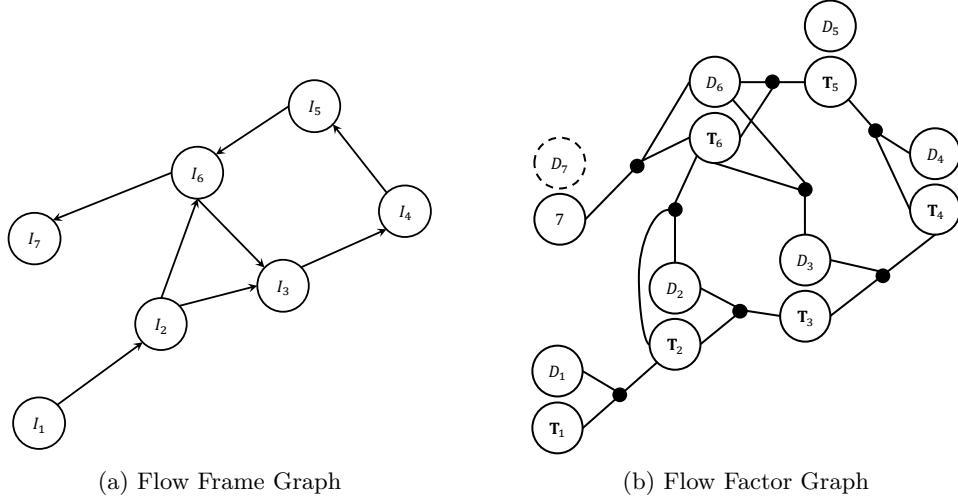


Figure 13.9 Flow frame graph and flow factor graph.

$\mathbf{z}_{ij}$  (because each edge in the frame graph gives rise to a single factor, we index the factors with the tuple  $(i, j)$  for notional convenience) and let the state variables  $\mathbf{x}$  be the poses  $(\mathbf{T}_1^w, \dots, \mathbf{T}_n^w)$  and dense depth maps  $(\mathbf{d}_1, \dots, \mathbf{d}_n)$ . We can construct a cost function over the states describing the flow that we should observe given the state variables

$$J(\mathbf{x}) \triangleq \sum_{(i,j) \in \mathcal{G}} \|\mathbf{z}_{ij} - \mathbf{h}_{ij}(\mathbf{T}_i^w, \mathbf{T}_j^w, \mathbf{d}_i)\|_{\Sigma_{ij}}^2 \quad (13.8)$$

where the *measurement prediction function*  $\mathbf{h}_{ij}$  was defined in (13.7).

This cost function gives rise to the factor graph in Figure 13.9b where each factor involves 3 state variables: the source pose, the destination pose and the source depth. A couple notes on parameterization: in practice, it is better to parameterize the depth variables using inverse depth because this leads to better local approximations of the cost function. Since poses are manifold state variables, we also want to use the local parameterization  $\exp(\xi_i^\wedge) \mathbf{T}_i^w$ .

### 13.3 Differentiable Bundle Adjustment and DROID-SLAM

We can use RAFT (or another off-the-shelf method) to compute optical flow, then minimize a least-squares optimization problem to solve for pose and depth. Alone, this doesn't work particularly well. Optical flow networks are trained on the task of optical flow, not 3D reconstruction or pose estimation. The loss function does not necessarily do a good job modeling what we care about in SLAM, particularly

the accuracy of the camera poses. The second issue is that the output of our flow network is subject to outliers and all pixels are weighted equally in the cost function.

*differentiable bundle adjustment*

One solution is to use intermediate representations, such as optical flow, as inputs to a differentiable optimization layer. Using the techniques outlined in Chapter 4, we can perform backpropagation through the optimization layer to supervise this intermediate representation and, by extension, train the weights of our network. Furthermore, we can use the network to output corresponding confidence weights to further shape the optimization landscape. To make this idea more concrete, recall in Chapter 1 that a factor graph gives rise to a cost function in the form

$$J(\mathbf{x}) \triangleq \sum_i \|\mathbf{e}_i(\mathbf{x}_i)\|_{\Sigma_i}^2 = \|\mathbf{e}(\mathbf{x})\|_{\Sigma}^2 \quad \mathbf{e}_i(\mathbf{x}) = \mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i) \quad (13.9)$$

where  $\mathbf{e}_i$  is the error vector representing the difference between the model of the world and the observed measurements. The error vector  $\mathbf{e}_i$  typically represents reprojection error (indirect methods) or photometric error (direct methods).

Instead of modeling the error directly, we can instead use a neural network to produce the error and corresponding confidence. We can define a neural network as a function  $\mathbf{e}_{\theta}$  which maps the current state  $\mathbf{x}$  (*e.g.*, poses) and measurements  $\mathbf{z}$  (*e.g.*, images) to an error vector  $\mathbf{e} \in \mathbb{R}^m$  and parametrize covariance in a similar way. We can rewrite the cost function plugging in the learned functions  $\mathbf{e}_{\theta}$  and  $\Sigma_{\theta}$

$$J(\mathbf{x}) \triangleq \|\mathbf{e}_{\theta}(\mathbf{z}, \mathbf{x})\|_{\Sigma_{\theta}(\mathbf{z}, \mathbf{x})}^2 \quad (13.10)$$

allowing the neural networks to shape the cost function.

BANet [1065] was among the first works to integrate BA into an end-to-end trainable architecture by defining a photometric error over feature maps extracted by a neural network. In this section, we outline DROID-SLAM [1079] and DPVO [1081] which build SLAM and VO systems using optical flow as an intermediate representation for differentiable BA.

### 13.3.1 DROID-SLAM Architecture

#### DROID-SLAM

The architecture of DROID-SLAM (Figure 13.10) reuses many of the components of RAFT. It takes in two inputs: the sequence of input images ( $I_1, \dots, I_n$ ) and a frame graph  $\mathcal{G}$  with edges between co-visible pairs of images. It outputs a sequence of pose and depth estimates.

**Feature Extraction and Visual Similarity.** The input to the network is a sequence of frames ( $I_1, \dots, I_n$ ) and a frame graph  $\mathcal{G}$ . Just like RAFT, DROID-SLAM uses a two feature extractors: a context encoder  $\mathbf{c}_{\theta}$  and an appearance encoder  $\mathbf{g}_{\theta}$ —both residual networks which output features at 1/8 the input resolution. The appearance features are used to build a set of correlation pyramids following RAFT. A correlation pyramid is constructed for each edge  $(i, j)$  in the frame graph using

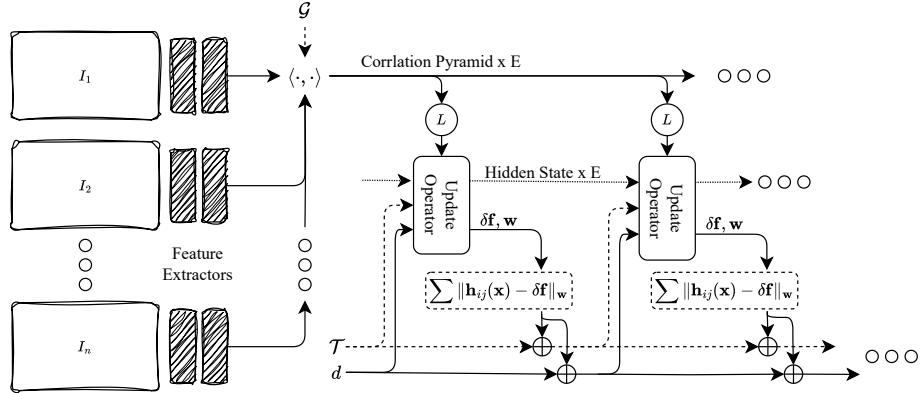


Figure 13.10 DROID-SLAM architecture: Features are extracted from the set of  $N$  input images and used to construct  $E$  correlation volumes (one for each edge in the frame graph). The update operator acts symmetrically on each edge in the frame graph. In each iteration, it uses the current estimate of the poses  $\mathbf{T}_1, \dots, \mathbf{T}_n$  and depths  $\mathbf{d}_1, \dots, \mathbf{d}_n$  to index from the correlation pyramid. The correlation features and hidden state are used to produce a flow revision  $\delta \mathbf{f}$  and confidence weights  $\mathbf{w}$  which parametrize the cost function. A couple Gauss-Newton updates are applied to generate pose and depth updates.

the correlation between appearance features of  $I_i$  and  $I_j$ . In total, if there are  $E$  edges in the frame graph,  $E$  correlation pyramids will be constructed.

**Update Operator.** The update operator acts on the frame graph  $G$  to produce global pose and depth updates. Each edge  $(i, j)$  in the frame graph has an associated correlation pyramid  $C_{ij}$  and hidden state  $q_{ij}$ . The hidden state  $q_{ij}$  is initialized from the context features of the source frame  $I_i$ .

Each iteration of the update operator uses the current estimate of poses and depth to compute the induced flow for each edge in the frame graph, using the measurement function defined in (13.7). Like RAFT, the update operator is a convolutional GRU, but here it is replicated for each edge in the frame graph. During each iteration, the update operator uses (1) the hidden state (2) correlation features (3) flow features and (4) the residual from the previous iteration to produce an updated hidden state. The updated hidden state is used to predict the measurement errors  $\delta \mathbf{f}_{ij}$  (e.g., flow field updates) and an associated confidence map  $\mathbf{w}_{ij}$ .

**Differentiable Bundle Adjustment.** The error estimate and the confidence map are used to construct a NLS cost function of the form BA

$$J(\mathbf{x}) \triangleq \sum_{(i,j) \in \mathcal{G}} \|\delta \mathbf{f}_{ij} - [\mathbf{h}_{ij}(\mathbf{x}_{ij}) - \mathbf{h}_{ij}(\mathbf{x}_{ij}^0)]\|_{\Sigma_{ij}}^2 \quad (13.11)$$

where the covariance is constructed by turning  $\mathbf{w}_{ij}$  into a block diagonal matrix  $\Sigma_{ij} = \text{diag } \mathbf{w}_{ij}$  and  $\mathbf{x}_{ij}^0$  are the initial pose and depth estimates at the start of the iteration. This cost function is saying that we want to update our states to drive

the change in the measurement function  $\mathbf{h}_{ij}(\mathbf{x}_{ij}) - \mathbf{h}_{ij}(\mathbf{x}_{ij}^0)$  to match the error  $\delta\mathbf{f}_{ij}$  as predicted by the network. The weights  $\mathbf{w}_{ij}$  give the network additional control of the optimization landscape by allowing it to place more weight on regions of the image where it is confident. The depth and camera pose are updated by unrolling Gauss Newton iterations. The techniques outlined in Chapter 4 are used to perform backpropagation through each individual Gauss Newton step allowing the network to be trained end-to-end.

**Trainings.** DROID-SLAM is trained on the synthetic TartanAir dataset [1163] which includes ground-truth depth and camera pose. Training involves a weighted combination of a *pose* and a *flow* loss. The *pose* loss penalizes the difference between the predicted poses of the network and the ground truth poses. Likewise, the *flow* loss penalizes the difference between the optical flow induced by the predicted pose and depth (13.6) and the optical flow induced by the ground-truth pose and depth.

### 13.3.2 DROID-SLAM Inference

The network is trained in an *in-place* setting, where it receives a fixed number of frames as input. There are several modifications that need to be made to operate in the online setting where the frame graph is dynamic and new frames are being constantly added and removed.

**Motion Filtering.** For initialization, it is important to select frames where there is sufficient motion. DROID-SLAM uses a simple probe to filter frames based on relative motion from the last keyframe. If  $I_{t-1}$  is the last keyframe and  $I_t$  is the incoming frame, it constructs a frame graph with one edge:  $I_t \rightarrow I_{t+1}$ , then runs the update operator to compute the flow revisions  $\delta\mathbf{f}$ . If the magnitude of  $\delta\mathbf{f}$  is below a threshold, the frame is discarded, otherwise it is kept.

**Initialization.** Once a sufficient number of frames have been accumulated, DROID-SLAM attempts to initialize. It constructs a frame graph where each frame is connected to its temporal neighbors, then runs several iterations of the update operator. Despite the simplicity of this initialization strategy, it works quite well on all benchmark datasets. Of course, this would fail in cases of pure rotation.

**Frontend.** Once a new frame is added, edges are added to the frame graph to include the new frame. Temporal edges are always added bidirectionally between the keyframe and the previous two frames. Induced optical flow is also used to compute the distance between all pairs of keyframes and new edges are added between frames with high co-visibility. DROID-SLAM takes steps to avoid adding too many edges such as applying non-maximal suppression to neighboring edges.

**Global Optimization.** DROID-SLAM also uses a backend process to optimize the full frame graph jointly. It first adds edges between temporally adjacent frames then uses the current estimate of poses and depths to compute optical flow between all pairs of frames. Long distance edges are added to the graph if the optical flow is below a certain threshold. The long distance edges enable min-range loop closure

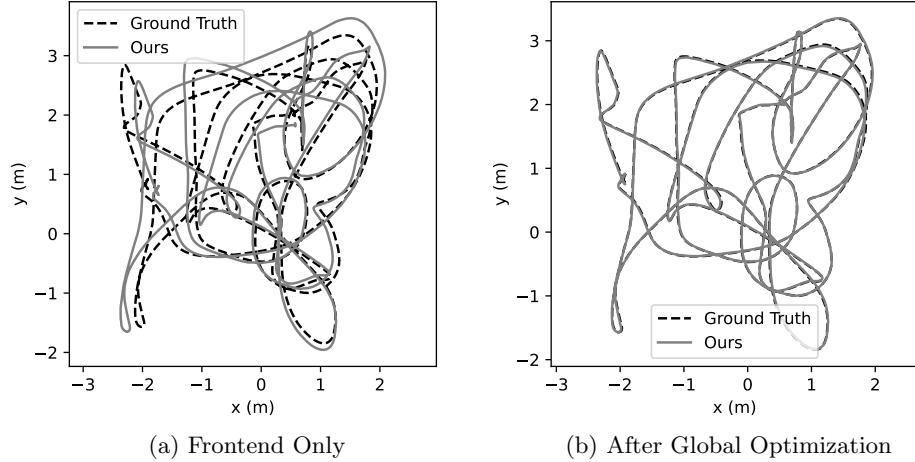


Figure 13.11 Estimated trajectory compared to ground truth on the `V1_02_medium` scene in the EuRoC benchmark. Global optimization reduces the rmse ATE from 16.5cm to 1.2cm.

assuming the drift isn't too severe but DROID-SLAM doesn't include any relocalization module, so large loops with lots of drift cannot be closed. As Figure 13.11 shows, global optimization with mid-range loop closure can significantly improve the accuracy of the estimated trajectory.

### 13.3.3 Generalizing to Other Modalities

The factor graph formulation makes it easy to generalize DROID-SLAM to RGB-D and stereo video without needing to retrain the network. In addition to RGB-D and stereo, this approach has also been generalized to visual-inertial SLAM [861] and event cameras [757].

**RGB-D Video.** With depth video, it is sufficient to simply add another factor encouraging the predicted depth to be close to the sensor depth. Sensor depth estimates are typically sparse, so this is only applied on pixels which have a valid sensor reading, producing the updated cost function

$$J(\mathbf{x}) \triangleq \sum_{(i,j) \in \mathcal{G}} \|\mathbf{z}_{ij} - \mathbf{h}_{ij}(\mathbf{x})\|^2 + \sum_i \|\bar{\mathbf{d}}_i - \mathbf{d}_i\|_{\Sigma_d}^2 \quad (13.12)$$

where  $\bar{\mathbf{d}}_i$  is the sensor depth observation and  $\Sigma_d$  is the sensor-covariance which is represented as a diagonal matrix with zeros on the missing sensor values.

**Stereo Video.** Generalizing to stereo videos is not much harder. In the stereo case, each input consists of a time-synchronized left and right frame taken from a

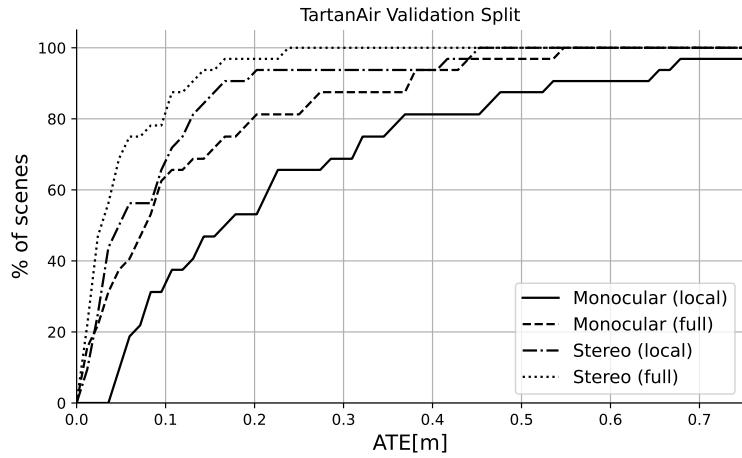


Figure 13.12 Comparison of monocular and stereo versions of DROID-SLAM with and without global optimization. DROID-SLAM is run all combinations on the 32 TartanAir validation sequences and plot the percentage of trajectories with absolute trajectory error (ATE) less than each threshold. Having stereo frames significantly helps—partially because it solves the problem of scale drift. Both the monocular and stereo system benefit from global optimization.

stereo rig. It is assumed that the relative pose between the left and the right frame is known. DROID-SLAM simply adds both the left and right frames to the frame graph and, for every left frame, an edge is added connecting it to its right pair. The relative pose between the left and right frame is fixed during optimization. In Figure 13.12, DROID-SLAM shows that stereo pairs of frames improve performance in addition to the performance benefit of using global optimization.

#### 13.3.4 Deep Patch Visual Odometry (DPVO)

DPVO

Deep Patch Visual Odometry [1081] is closely related to DROID-SLAM but instead operates on sparse *patches* instead of dense frames. This turns out to have several advantages, not just for computational efficiency, but also accuracy.

**Patch Graph.** DPVO operates on patches instead of full frames and uses a *patch graph* data structure to store the association between patches and images. The patch graph is a bipartite graph representing the patch lifetimes; edges in the graph connect patches with frames.

Each patch is represented as a fronto-parallel plane in the image from which it was extracted. Thus, each patch is parameterized by a single (inverse) depth. To

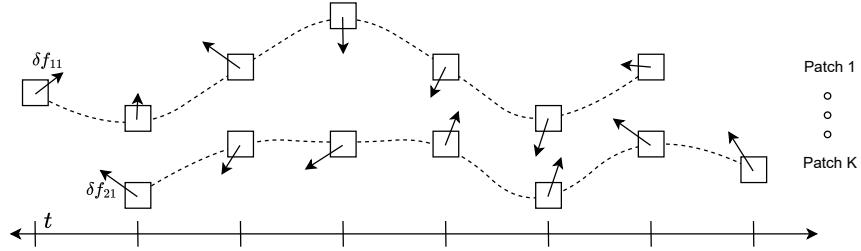


Figure 13.13 DPVO patch graph: each patch is tracked through time. The update operator outputs flow revisions  $\delta f$  for each edge in the graph, denoted by the error vector in the figure.

reproject patches between frames, we update the measurement function

$$\mathbf{h}_{ij}(\mathbf{T}_{s(i)}^w, \mathbf{T}_j^w, d_i) = \pi(\mathbf{T}_{s(i)}^j \cdot \pi^{-1}(\mathbf{u}, \mathbf{v}, d_i)) - \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \quad (13.13)$$

where  $i$  is the patch index and  $s(i)$  is the source index, or the index of the frame from which patch  $i$  was extracted.  $\mathbf{u}$  and  $\mathbf{v}$  are the pixel coordinates of the patch and  $d_i$  is a single scalar depth, shared over the full patch.

**Patch Extraction.** Each incoming image is processed by a feature encoder and a context encoder which outputs features at 1/4 resolution. Patch centroids are randomly sampled and used to construct a  $3 \times 3$  patch with one pixel offsets (1px in the resolution of the extracted feature maps so actually this corresponds to 4px in the original image). DPVO uses a very simple method for constructing the patch graph. Each patch is connected to all the frames within a radius of  $r$  timesteps. Computation can be traded off for accuracy by selecting more patches and increasing their lifetime.

**Update Operator.** The update operator acts on edges in the patch graph. Like RAFT and DROID-SLAM, the update operator used by DPVO is a recurrent weight-tied unit but incorporates several new components. It performs temporal convolutions across the patch lifetime in addition to aggregation layers which pool features across frames and patches. The update operator produces a per-edge flow update  $\delta f$  and accompanying confidence weights  $w$ . These are used to construct a cost function using (13.7) as the measurement function and the pose and depth updates are produced by unrolling two GN iteration.

**Inference.** DPVO starts by accumulating frames for initialization. Once a set number of frames is accumulated, DPVO initializes by running several iterations of the update operator. Subsequently, when each new frame is added, patches are extracted from the new frame and edges are added to the patch graph—both forward edges connecting previous patches with the new frame and backward edges con-

necting the new patches with previous frames. Next, the update operator is run to refine the depth and pose estimates. Like DROID-SLAM, DPVO also implements keyframing to ensure that the optimization window is sufficiently large.

**Loop Closure and Global Optimization.** DPVO is purely a visual odometry system and cannot perform loop closures outside the optimization window. Deep Patch Visual SLAM (DPVS) [669] expands on DPVO by adding global optimization and loop closure. DPVS can perform long-range loop closures using a relocalization module and short-range loop closures based on proximity detection. Unlike DROID-SLAM, DPVS can maintain real-time performance on a single GPU.

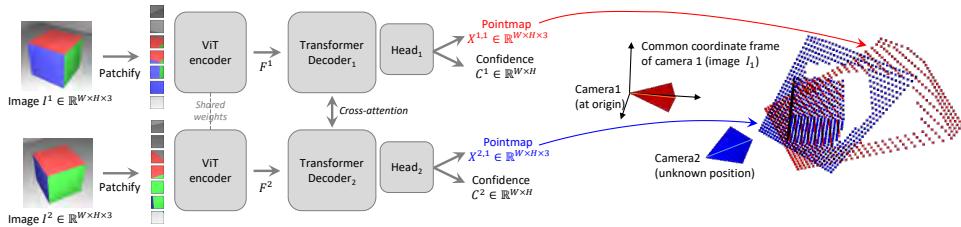
### 13.4 DuSt3R

As shown in the previous sections, modern SLAM systems can be described from a high-level point of view as pipelines involving a series of steps: first, obtaining pixel correspondences between frames, involving keypoints, patches, or dense optical flow; then, modeling the factor graph associated between the state variable  $\mathbf{x}$  (*e.g.*, camera parameters and depth maps) and the observed correspondences; and finally solving for the state variable update that would minimize the overall cost function  $J(\mathbf{x})$ . Note that, in the larger context of SFM, camera parameters to be estimated would also include the intrinsic calibration of each camera, while those are generally assumed to be given in the context of SLAM. In previous chapters, we have seen how learning-based approaches can be inserted within each of these steps (*e.g.*, neural network for estimating the optical flow), but the SLAM pipeline remains heavily handcrafted and rather complex overall.

DuSt3R

Recently, a novel type of learning-based approach has emerged. Instead of breaking down the SLAM problem as a series of atomic steps, DUSt3R [1161] proposes instead to let a neural network solve all steps at once: solely given images, *it directly outputs camera poses, dense depth maps, and even intrinsic parameters*, without requiring any explicit solvers nor iterative algorithms. Note that applications for this type of network go well beyond SLAM, as it additionally makes no assumption on the temporal relation between the input images, and therefore more generally applies to SFM and dense 3D reconstruction.

In reality, however, tasking a network with reconstructing a large scene from possibly thousands of frames still seems out of reach, and in practice, DUSt3R only solves a *local* version of the 3D reconstruction problem involving just two input images. As we will see later, in the end, a form of factor graph is still necessary for performing SLAM (or large-scale 3D reconstruction) given a set of local 3D reconstructions. The difference with previous methods is that, due to the richness of its output, DUSt3R simplifies and expands the possibilities of SLAM systems, making them far more robust and enabling new features like handling varying camera intrinsics (*e.g.*, dynamic zoom-in / zoom-out).



**Figure 13.14 Architecture of the network.** Two views of a scene ( $I^1, I^2$ ) are first encoded in a Siamese manner with a shared ViT encoder. The resulting token representations  $F^1$  and  $F^2$  are then passed to two transformer decoders that constantly exchange information via cross-attention. Finally, two regression heads output the two corresponding pointmaps and associated confidence maps. Importantly, the two pointmaps are expressed in the same coordinate frame of the first image  $I^1$ . The network is trained using a simple regression loss (Eq. (13.17)). Image from [1161] (©2024 IEEE).

**Overview.** In a nutshell, DUS3R is able to perform Dense Unconstrained Stereo 3D Reconstruction from *un-calibrated* and *un-posed* cameras. Its main component is a network that can regress a dense and accurate scene representation solely from a *pair* of images, without prior information regarding the scene nor the cameras (not even the intrinsic parameters). The resulting scene representation is based on *3D pointmaps* with rich properties that simultaneously encapsulate (a) the scene geometry, (b) the relation between pixels and scene points and (c) the relation between the two viewpoints. From this output alone, practically all scene parameters (*i.e.*, cameras and scene geometry) can be straightforwardly extracted. This is possible because the network jointly processes the input images and the resulting 3D pointmaps, thus learning strong geometric and shape priors which are reminiscent of those commonly leveraged in multi-view stereo, like shape from texture, shading, or contours [994].

**Pointmap.** We denote a dense 2D field of 3D points as a *pointmap*  $\mathbf{X} \in \mathbb{R}^{W \times H \times 3}$ . In association with its corresponding RGB image  $I$  of resolution  $W \times H$ ,  $\mathbf{X}$  forms a one-to-one mapping between image pixels and 3D scene points, *i.e.*,  $I_{i,j} \leftrightarrow \mathbf{X}_{i,j}$ , for all pixel coordinates  $(i, j) \in \{1 \dots W\} \times \{1 \dots H\}$ . We assume here that each camera ray hits a single 3D point, *i.e.*, ignoring the case of translucent surfaces.

Given camera intrinsics  $\mathbf{K} \in \mathbb{R}^{3 \times 3}$ , the pointmap  $\mathbf{X}$  of the observed scene can be straightforwardly obtained from the ground-truth depth map  $\mathbf{D} \in \mathbb{R}^{W \times H}$  as  $\mathbf{X}_{i,j} = \mathbf{K}^{-1} \mathbf{D}_{i,j} [i, j, 1]^\top$ . Here,  $\mathbf{X}$  is expressed in the camera coordinate frame. In the following, we denote as  $\mathbf{X}^{n,m}$  the pointmap  $\mathbf{X}^n$  from camera  $n$  expressed in camera  $m$ 's coordinate frame:

$$\mathbf{X}^{n,m} = h^{-1} (\mathbf{P}_m \mathbf{P}_n^{-1} h (\mathbf{X}^n)) \quad (13.14)$$

where  $\mathbf{P}_m, \mathbf{P}_n \in \text{SE}(3)$  are the world-to-camera poses for images  $m$  and  $n$ , the func-

tion  $h : (x, y, z) \rightarrow (x, y, z, 1)$  is the homogeneous mapping, and  $h^{-1} : (x, y, z, 1) \rightarrow (x, y, z)$  is its inverse mapping.

### 13.4.1 A Network for Generalized Stereo Reconstruction

**Problem definition.** We build a network that solves the 3D reconstruction task for the generalized stereo case through direct regression. The network  $f$  takes as input two RGB images  $I^1, I^2 \in \mathbb{R}^{W \times H \times 3}$  and outputs two corresponding pointmaps  $\mathbf{X}^{1,1}, \mathbf{X}^{2,1} \in \mathbb{R}^{W \times H \times 3}$  with associated confidence maps  $\mathbf{C}^{1,1}, \mathbf{C}^{2,1} \in \mathbb{R}^{W \times H}$ . Note that both pointmaps are expressed in the *same* coordinate frame of  $I^1$ , which radically differs from existing approaches but offers multiple key advantages (see 13.4.3).

**Network architecture.** As shown in Figure 13.14, the architecture of the network is composed of two identical branches (one for each image) comprising each an image encoder, a decoder and a regression head. The two input images are first encoded in a Siamese manner by the same weight-sharing ViT encoder [280], yielding two token representations  $F^1$  and  $F^2$ :

$$F^1 = \text{Encoder}(I^1), \quad F^2 = \text{Encoder}(I^2).$$

The network then reasons over both of them jointly in the decoder, a generic transformer network equipped with cross attention. Each decoder block thus sequentially performs self-attention (each token of a view attends to tokens of the same view), then cross-attention (each token of a view attends to all other tokens of the other view), and finally feeds tokens to a MLP. Importantly, information is constantly shared between the two branches during the decoder pass. This is crucial in order to output properly aligned pointmaps. Namely, each decoder block attends to tokens from the other branch:

$$\begin{aligned} G_i^1 &= \text{DecoderBlock}_i^1(G_{i-1}^1, G_{i-1}^2), \\ G_i^2 &= \text{DecoderBlock}_i^2(G_{i-1}^2, G_{i-1}^1), \end{aligned}$$

for  $i = 1, \dots, B$ , for a decoder with  $B$  blocks and initialized with encoder tokens  $G_0^1 := F^1$  and  $G_0^2 := F^2$ . Here,  $\text{DecoderBlock}_i^v(G^1, G^2)$  denotes the  $i$ -th block in branch  $v \in \{1, 2\}$ ,  $G^1$  and  $G^2$  are the input tokens, with  $G^2$  the tokens from the other branch. Finally, in each branch a separate regression head takes the set of decoder tokens and outputs a pointmap and an associated confidence map:

$$\begin{aligned} \mathbf{X}^{1,1}, \mathbf{C}^{1,1} &= \text{Head}^1(G_0^1, \dots, G_B^1), \\ \mathbf{X}^{2,1}, \mathbf{C}^{2,1} &= \text{Head}^2(G_0^2, \dots, G_B^2). \end{aligned}$$

It should be noted that this generic architecture never explicitly enforces any geometrical constraints. Hence, pointmaps do not necessarily correspond to any physically plausible camera model (but they closely fit in practice). Rather, we

let the network learn all relevant priors present from the train set, which only contains geometrically consistent pointmaps. Using a generic architecture allows to leverage strong pretraining techniques, ultimately surpassing what existing task-specific architectures can achieve.

#### 13.4.2 Regression Loss and Confidence-Aware Loss

**3D Regression loss.** The sole training objective is based on regression in the 3D space. Let us denote the ground-truth pointmaps as  $\hat{\mathbf{X}}^{1,1}$  and  $\hat{\mathbf{X}}^{2,1}$ , obtained from Eq. (13.14) along with two corresponding sets of valid pixels  $\mathcal{D}^1, \mathcal{D}^2 \subseteq \{1 \dots W\} \times \{1 \dots H\}$  for which the ground-truth is defined. The regression loss for a valid pixel  $i \in \mathcal{D}^v$  in view  $v \in \{1, 2\}$  is simply defined as the Euclidean distance:

$$\ell_{\text{regr}}(v, i) = \left\| \frac{1}{z} \mathbf{X}_i^{v,1} - \frac{1}{\bar{z}} \hat{\mathbf{X}}_i^{v,1} \right\|. \quad (13.15)$$

To handle the scale ambiguity between prediction and ground-truth, we normalize the predicted and ground-truth pointmaps by scaling factors  $z = \text{norm}(\mathbf{X}^{1,1}, \mathbf{X}^{2,1})$  and  $\bar{z} = \text{norm}(\hat{\mathbf{X}}^{1,1}, \hat{\mathbf{X}}^{2,1})$ , respectively, which simply represent the average distance of all valid points to the origin:

$$\text{norm}(\mathbf{X}^1, \mathbf{X}^2) = \frac{1}{|\mathcal{D}^1| + |\mathcal{D}^2|} \sum_{v \in \{1, 2\}} \sum_{i \in \mathcal{D}^v} \|\mathbf{X}_i^v\|. \quad (13.16)$$

**Confidence-aware loss.** In reality, there are ill-defined 3D points, *e.g.*, in the sky or on translucent objects. More generally, some parts in the image are typically harder to predict than others. We thus jointly learn to predict a score for each pixel which represents the confidence that the network has about this particular pixel. The final training objective is the *confidence-weighted regression loss* from Eq. (13.15) over all valid pixels:

$$\mathcal{L}_{\text{conf}} = \sum_{v \in \{1, 2\}} \sum_{i \in \mathcal{D}^v} \mathbf{C}_i^{v,1} \ell_{\text{regr}}(v, i) - \alpha \log \mathbf{C}_i^{v,1}, \quad (13.17)$$

where  $\mathbf{C}_i^{v,1}$  is the confidence score for pixel  $i$ , and  $\alpha$  is a hyper-parameter controlling the regularization term [221]. To ensure a strictly positive confidence, we typically define  $\mathbf{C}_i^{v,1} = 1 + \exp c_i^{v,1} \gg 0$ , with  $c_i^{v,1} \in \mathbb{R}$ . This has the effect of forcing the network to extrapolate in harder areas, *e.g.*, those ones covered by a single view. Training network  $f$  with this objective allows to estimate confidence scores without an explicit supervision. Examples of input image pairs with their corresponding outputs are shown in Figure 13.15.

#### 13.4.3 Downstream Applications

Rich properties of the output pointmaps allow us to recover all scene parameters as described below.

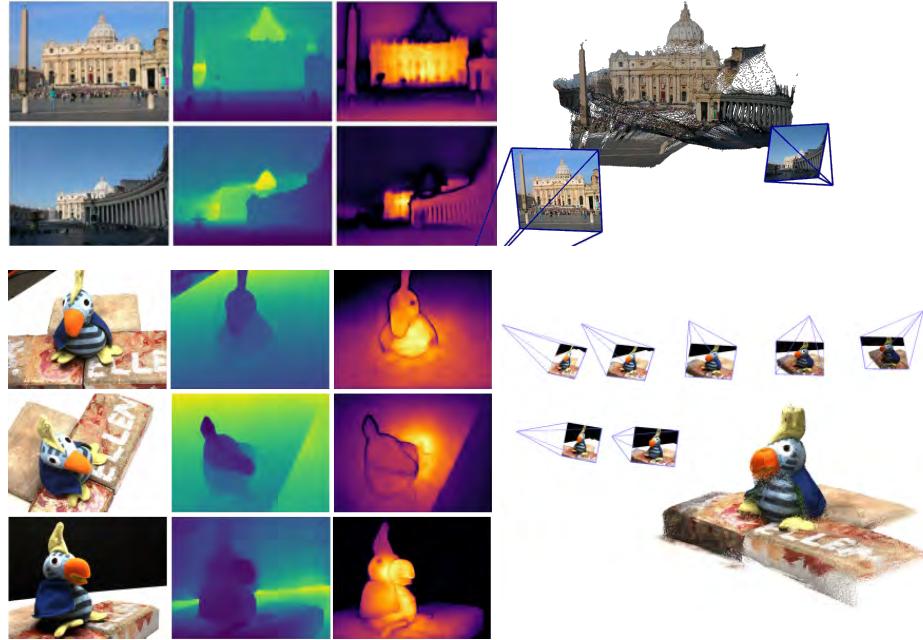


Figure 13.15 **Reconstruction examples** on two scenes never seen during training. From left to right: RGB, depth map, confidence map, reconstruction. The top scene shows the raw result output from  $f(I^1, I^2)$ . The bottom scene shows the outcome of global alignment (Sec. 13.4.4) Image from [1161] (©2024 IEEE).

**Recovering intrinsics.** By definition, the pointmap  $\mathbf{X}^{1,1}$  is expressed in  $I^1$ 's coordinate frame. It is therefore possible to estimate the camera intrinsic parameters by solving a simple optimization problem. We assume that the principal point is approximately centered and pixels are squares, hence only the focal length  $f_1^*$  remains to be estimated:

$$f_1^* = \arg \min_{f_1} \sum_{i=0}^W \sum_{j=0}^H \mathbf{C}_{i,j}^{1,1} \left\| (i', j') - f_1 \frac{(\mathbf{X}_{i,j,0}^{1,1}, \mathbf{X}_{i,j,1}^{1,1})}{\mathbf{X}_{i,j,2}^{1,1}} \right\|,$$

with  $i' = i - \frac{W}{2}$  and  $j' = j - \frac{H}{2}$ . Fast iterative solvers, *e.g.*, based on the Weiszfeld algorithm [875], can find the optimal  $f_1^*$  in a few iterations. For the focal  $f_2^*$  of the second camera, the simplest option is to perform the inference for the pair  $(I^2, I^1)$  and use above formula with  $\mathbf{X}^{2,2}$  instead of  $\mathbf{X}^{1,1}$ .

**Relative pose estimation.** One way is to perform 2D matching and recover intrinsics as described above, then estimate the Epipolar matrix and recover the relative pose [437]. Another, more direct way is to compare the pointmaps  $\mathbf{X}^{1,1} \leftrightarrow \mathbf{X}^{1,2}$  (or, equivalently,  $\mathbf{X}^{2,2} \leftrightarrow \mathbf{X}^{1,2}$ ) using Procrustes alignment [705] to get the scaled relative pose  $\mathbf{P}^* = \sigma^*[\mathbf{R}^* | \mathbf{t}^*]$ :

$$\mathbf{P}^* = \arg \min_{\sigma, \mathbf{R}, \mathbf{t}} \sum_i \mathbf{C}_i^{1,1} \mathbf{C}_i^{1,2} \left\| \sigma(\mathbf{R} \mathbf{X}_i^{1,1} + \mathbf{t}) - \mathbf{X}_i^{1,2} \right\|^2,$$

which can be achieved in closed-form. Procrustes alignment is, unfortunately, sensitive to noise and outliers. A more robust solution is to rely on RANSAC [328] with PnP [437, 638].

#### 13.4.4 Global Alignment

The network  $f$  presented in the previous section can only handle a pair of images. DUSt3R [1161] also proposed a simple post-processing optimization for larger scenes. It enables the alignment of pointmaps predicted from multiple images into a joint 3D space. This is possible thanks to the rich content of the pointmaps, which encompasses by design two aligned point-clouds and their corresponding pixel-to-3D mapping.

**Pairwise graph.** Given a set of images  $\{I^1, I^2, \dots, I^N\}$  for a given scene, we first construct a connectivity graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  where  $N$  images form vertices  $\mathcal{V}$  and each edge  $e = (n, m) \in \mathcal{E}$  indicates that images  $I^n$  and  $I^m$  share some visual content. To that aim, we either use existing off-the-shelf image retrieval methods, or we pass all pairs through network  $f$  (inference takes  $\approx 25$ ms on a H100 GPU) to measure their overlap from the average confidence in both pairs, and then filter out low-confidence pairs.

**Global optimization.** We use the connectivity graph  $\mathcal{G}$  to recover *globally aligned* pointmaps  $\{\chi^n \in \mathbb{R}^{W \times H \times 3}\}$  for all cameras  $n = 1 \dots N$ . To that aim, we first predict, for each image pair  $e = (n, m) \in \mathcal{E}$ , the pairwise pointmaps  $\mathbf{X}^{n,n}, \mathbf{X}^{m,n}$  and their associated confidence maps  $\mathbf{C}^{n,n}, \mathbf{C}^{m,n}$ . For the sake of clarity, let us define  $\mathbf{X}^{n,e} := \mathbf{X}^{n,n}$  and  $\mathbf{X}^{m,e} := \mathbf{X}^{m,n}$ . Since the goal involves to express all pairwise predictions in a common coordinate frame, we introduce a pairwise pose  $\mathbf{P}_e \in \mathbb{R}^{3 \times 4}$  and scaling  $\sigma_e > 0$  associated to each pair  $e \in \mathcal{E}$ . We then formulate the following optimization problem:

$$\chi^* = \arg \min_{\mathbf{x}, \mathbf{P}, \sigma} \sum_{e \in \mathcal{E}} \sum_{v \in e} \sum_{i=1}^{HW} \mathbf{C}_i^{v,e} \|\chi_i^v - \sigma_e \mathbf{P}_e \mathbf{X}_i^{v,e}\|. \quad (13.18)$$

Here, with some abuse of notation, we write  $v \in e$  for  $v \in \{n, m\}$  if  $e = (n, m)$ . The idea is that, for a given pair  $e$ , the *same* rigid transformation  $P_e$  should align both pointmaps  $\mathbf{X}^{n,e}$  and  $\mathbf{X}^{m,e}$  with the world-coordinate pointmaps  $\chi^n$  and  $\chi^m$ , since  $\mathbf{X}^{n,e}$  and  $\mathbf{X}^{m,e}$  are by definition both expressed in the same coordinate frame. To avoid the trivial optimum where  $\sigma_e = 0, \forall e \in \mathcal{E}$ , we enforce that  $\prod_e \sigma_e = 1$ .

**Recovering camera parameters.** A straightforward extension to this framework enables to recover all cameras parameters. By simply replacing

$$\chi_{i,j}^n := \mathbf{P}_n^{-1} h(\mathbf{K}_n^{-1} \mathbf{D}_{i,j}^n [i, j, 1]^\top)$$

(*i.e.*, enforcing a standard camera pinhole model as in Eq. (13.14)), we can thus estimate all camera poses  $\{\mathbf{P}_n\}$ , associated intrinsics  $\{\mathbf{K}_n\}$  and depthmaps  $\{\mathbf{D}^n\}$  for  $n = 1 \dots N$ .

### 13.5 MAST3R

Two-view 3D reconstruction priors, pioneered by DUST3R [1161] has created a paradigm shift in structure from motion (SFM) by capitalizing on curated 3D datasets. Beyond intrinsics and relative poses, DUST3R can also obtain reliable pixel correspondences from pointmaps, by looking for reciprocal matches in some invariant feature space [1161, 295, 987, 1190]. While such a scheme works well even in the presence of extreme viewpoint changes, the resulting correspondences are rather imprecise, yielding suboptimal accuracy. This is a rather natural result as (i) regression is inherently affected by noise, and (ii) because DUST3R was never explicitly trained for matching.

#### 13.5.1 Matching Head

*MAS3R* For these reasons, MAS3R proposed to extend DUST3R by adding a second head that outputs two dense feature maps  $\mathbf{D}^1$  and  $\mathbf{D}^2 \in \mathbb{R}^{H \times W \times d}$  of dimensionality  $d$ :

$$\mathbf{D}^1 = \text{Head}_{\text{desc}}^1([H^1, H'^1]), \quad (13.19)$$

$$\mathbf{D}^2 = \text{Head}_{\text{desc}}^2([H^2, H'^2]). \quad (13.20)$$

The head is implemented as a simple 2-layers MLP interleaved with a non-linear GELU activation function [447]. Each local feature is normalized to have unit norm.

#### 13.5.2 Matching Objective

MAS3R's matching objective is to encourage each local descriptor from one image to match with at most a single descriptor from the other image that represents the same 3D point in the scene. To that aim, MAS3R leverages the infoNCE [1117] loss over the set of ground-truth correspondences  $\hat{\mathcal{M}} = \{(i, j) | \hat{\mathbf{X}}_i^{1,1} = \hat{\mathbf{X}}_j^{2,1}\}$ :

$$\mathcal{L}_{\text{match}} = - \sum_{(i,j) \in \hat{\mathcal{M}}} \log \frac{s_\tau(i, j)}{\sum_{k \in \mathcal{P}^1} s_\tau(k, j)} + \log \frac{s_\tau(i, j)}{\sum_{k \in \mathcal{P}^2} s_\tau(i, k)}, \quad (13.21)$$

$$\text{with } s_\tau(i, j) = \exp [-\tau \mathbf{D}_i^{1\top} \mathbf{D}_j^2]. \quad (13.22)$$

Here,  $\mathcal{P}^1 = \{i | (i, j) \in \hat{\mathcal{M}}\}$  and  $\mathcal{P}^2 = \{j | (i, j) \in \hat{\mathcal{M}}\}$  denote the subset of considered pixels in each image and  $\tau$  is a temperature hyper-parameter. Note that this matching objective is essentially a cross-entropy *classification* loss: contrary to the regression loss in DUST3R the network is only rewarded if it gets the correct pixel



Figure 13.16 Qualitative example of MUSt3R reconstructions of Aachen Day-Night [1281] nexus4 sequence 5 (offline, *top*) and TUM-RGBD [1042] room sequence (online, *bottom*). Images from [135] (©2025 IEEE).

right, not a nearby pixel. This strongly encourages the network to achieve high-precision matching. Finally, both regression and matching losses are combined to get the final training objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{conf}} + \beta \mathcal{L}_{\text{match}}. \quad (13.23)$$

### 13.6 Extending MASt3R to SfM and SLAM

DUSt3R and MASt3R are designed to estimate pointmaps between pairs of images. As we saw, global optimization can extend MASt3R and DUSt3R to produce multi-frame reconstructions. In this section, we explore works which integrate MASt3R into full SfM and SLAM systems which are capable of reconstruction full video sequences and large collections of frames.

#### 13.6.1 MASt3R-SfM

As MASt3R [640] is able to perform local 3D reconstruction and matching in a single forward pass, MASt3R-SfM [289] proposed to extended it to a fully-integrated SfM pipeline that can handle completely unconstrained input image collections, *i.e.*, ranging from a single view to large-scale scenes, possibly without any camera motion. Since MASt3R is fundamentally limited to processing image pairs, it scales poorly to large image collections. To address this issue, MASt3R-SfM modified its frozen encoder to perform fast image retrieval with negligible computational overhead, resulting in a scalable SfM method with quasi-linear complexity in the number of images. Thanks to the robustness of MASt3R to outliers, the proposed method is able to completely get rid of RANSAC. The SfM optimization is carried out in two successive gradient descents based on frozen local reconstructions output by MASt3R: first, using a matching loss in 3D space; then with a 2D reprojection loss to refine the previous estimate.

*MASt3R-SfM*

### 13.6.2 MUSt3R

MUSt3R-SfM works seamlessly with dozens of images, but when feeding even many images, the pairwise nature of DUSt3R and MASt3R becomes a drawback rather than a strength. Since the predicted pointmaps are expressed in a local coordinate system defined by the first image of each pair, all predictions live in different coordinate systems. This design hence requires a global post-processing step to align all predictions into one global coordinate frame, which quickly becomes intractable for large collections when done naively. This raises the problem of the quadratic complexity of a pairwise approach and a robust and quick optimization in the real-time scenario. While these concerns are partially addressed in MASt3R-SfM [289], MUSt3R [135] takes a different stance and designs a new architecture that is scalable to large image collections of arbitrary scale, and that can infer the corresponding pointmaps in the same coordinate system at high frame-rates. To achieve this, MUSt3R extends the DUSt3R architecture through several crucial modifications—*i.e.*, making it symmetric and adding a working memory mechanism—with limited added complexity. The model, beyond handling *offline* reconstruction of unordered image collections in a SFM scenario, can also tackle the task of dense visual odometry (VO) and SLAM, which aims to predict *online* the camera pose and 3D structure of a video stream recorded by a moving camera. MUSt3R can seamlessly leverage the memory mechanism to cover both scenarios such that no architecture change is required and the same network can operate either task in an agnostic manner (see Figure 13.16).

MUSt3R

MASt3R-SLAM

### 13.6.3 MASt3R-SLAM

We compete this chapter by presenting an alternative approach to make MASt3R scalable. MASt3R-SLAM [786] is a real-time SLAM framework to leverage MASt3R’s two-view 3D reconstruction priors as a unifying foundation for tracking, mapping, and relocalization, as shown in Figure 13.17. While MASt3R-SfM has applied these priors to SFM in an offline setting with unordered image collections [289], SLAM receives data incrementally and must maintain real-time operation that requires solutions on low-latency matching, careful map maintenance, and efficient methods for large-scale optimization. On one side, MASt3R-SLAM adopts filtering and optimization techniques from SLAM systems [1077, 1078], and performs local filtering of pointmaps in the frontend to enable large-scale global optimization in the backend. On the other hand, like in MASt3R, it makes no assumption on each image’s camera model beyond having a unique camera center that all rays pass through. This results in a real-time dense monocular SLAM system capable of reconstructing scenes with generic, time-varying camera models. Given calibration, MASt3R-SLAM demonstrated state-of-the-art performance in trajectory accuracy and dense geometry estimation.

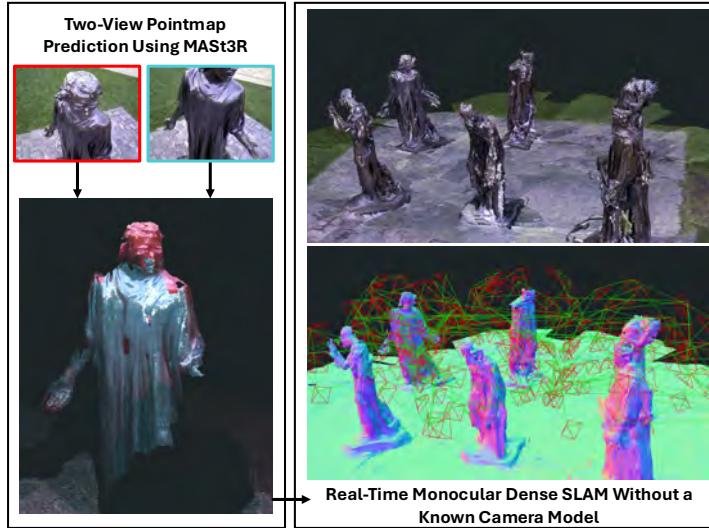


Figure 13.17 Reconstruction from the dense monocular SLAM system on the Burghers sequence [1293]. Using the two-view predictions from MAST3R shown on the left, the system achieves globally reconstructions in real-time without a known camera model [786] (©2025 IEEE).

### 13.7 Further Readings & Recent Trends

Deep SLAM systems have been progressing along multiple different paths. There are ongoing efforts to improve the scalability of deep SLAM systems in addition to further advancing their accuracy and robustness.

One line of research has continued scaling feed forward networks such as DUSt3R to hundreds of images [1153, 1222]. VGGT [1153] can reconstruct hundreds of images in the order of seconds and predicts camera pose, intrinsics, depths, pointmaps and feature tracks directly without the need for 3D optimization and post-processing. VGGT, however, still benefits from using the predicted features tracks to further refine camera poses and intrinsics using bundle adjustment, so factor graphs still play an essential role when highly accurate predictions are necessary. To use VGGT for longer sequences of images, VGGT-SLAM [721] builds a SLAM framework where local submaps are reconstructed with VGGT and aligned into a global coordinate frame using a factor graph.

Another line of work has continued to refine the combination of factor graphs with deep learning models. While DROID-SLAM requires camera calibration to be known at input and fails in moderately dynamic environments, MegaSAM [658] and Video Pose Engine (ViPE) [489] can operate in highly dynamic environments and do not require known calibration. Instead they jointly optimize camera calibration during inference. These methods leverage the latest depth estimation and semantic

segmentation models and rely on factor graphs to fuse predictions from multiple models.

A recent trend of SLAM and 3D reconstruction models has been to build on top of pretrained vision transformers. Vision foundation models trained using self-supervised learning techniques [1174, 828] (see also Chapter 17) have proved to provided excellent initialization for downstream 3D reconstruction tasks. As pre-training methods improve, these improvements may translate into improvements on downstream 3D tasks.

The culmination of these developments are leading to systems that can seamlessly handle challenging videos featuring dynamic objects, pure rotation, and unknown camera calibration. Recent methods leveraging deep learning have proved robust enough to reconstruct internet video at scale, such as 100k YouTube videos [489, 928].

# 14

## Map Representations with Differentiable Volume Rendering

Hidenobu Matsuki and Andrew J. Davison

The choice of underlying scene representation significantly impacts the capability and efficiency of SLAM systems. As discussed in Chapter 5, each scene or map representation in SLAM has its own advantages and disadvantages, depending on the specific downstream task, such as camera ego-motion tracking, dense scene reconstruction, or object-level decomposition. Recently, 3D scene representations that can be optimized via differentiable rendering, such as Neural radiance field (NeRF) [761] or 3D Gaussian Splatting (3DGS) [551], have emerged as prominent choices for high-quality inverse rendering and novel view synthesis within the computer graphics research community, driven by advances in graphics hardware and automatic differentiation frameworks.

Initially focused on offline novel view synthesis with known camera poses, these representations have gradually been applied to various other tasks in 3D reconstruction and scene understanding [1199], such as accurate and dense geometry reconstruction of large-scale scenes [816, 1238, 1156], semantic fusion [1288, 1289, 585], and more. Early implementations required days of optimization, but by leveraging explicit data structures (*e.g.*, voxels or point clouds) to enable fast data access, optimization times have fallen to near real-time. More recently, 3D Gaussian Splatting has been introduced, offering super-fast and high-quality rendering capabilities, effectively replacing Neural Radiance Fields in many applications. These novel graphics primitives are now also being integrated into SLAM research. Although the idea of combining differentiable rendering with 3D occupancy maps dates back to Thrun et al.’s idea of “forward sensor model” from the early 2000s [1086], modern hardware and algorithms are opening entirely new possibilities.

This chapter provides an overview of NeRF and 3DGS as scene representations for SLAM, discussing their advantages and disadvantages, and suggesting future research directions. We begin with a historical survey of 3D scene representations (Section 14.1), explore NeRF (Section 14.2) and 3DGS (Section 14.3) in detail, including their integration into SLAM. We then conclude the chapter by discussing potential future directions (Section 14.4).

## 14.1 3D Scene Representation and Differentiable Rendering

### 14.1.1 Learnable 3D Representations

With the success of deep learning in the 2D image domain [260, 443, 444], the research community gradually started to explore learning-based approaches for 3D tasks. Early works directly utilized 3D data represented by classical data structures such as points [887] and occupancy grids [926], processing them for downstream tasks. However, these classical representations are discrete and often not amenable for deep learning; they often struggle to represent watertight and continuous surfaces with flexible topology in a compact manner, which limits the gradient-based optimization essential for neural network training. To create more optimization-friendly 3D representations, some early works introduced 3D representations using neural networks as scalar function approximators to define occupancy or signed distance functions [756, 844, 192], which can be trained through direct 3D supervision. DeepSDF [844], for example, directly regresses a signed distance function (SDF) from a 3D coordinate and optionally a latent code using 8-layer Multi-Layer Perceptron (MLP) as shown in Figure 14.1. These methods demonstrated continuous and compact 3D shape reconstruction without being constrained by topology and discretization errors. However, supervised training on 3D data requires costly ground-truth data annotations, making dataset collection non-scalable. This limitation has motivated research efforts toward leveraging easier-to-obtain 2D image information and varying levels of supervision for 3D scene understanding, which has led to 3D reconstruction via differentiable rendering.

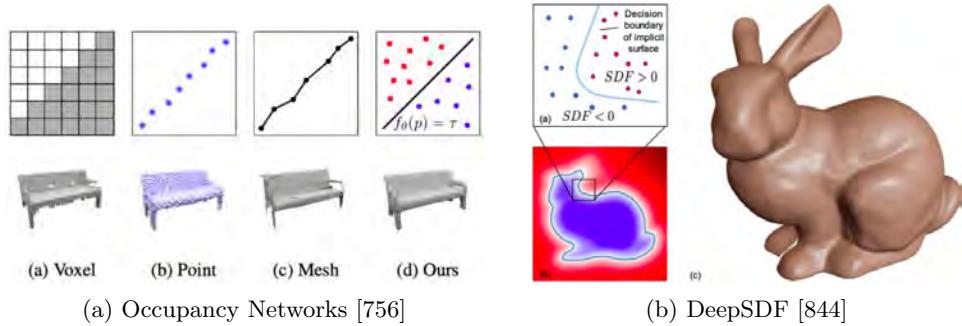


Figure 14.1 Learnable 3D Representations. Both ONet [756] (©2019 IEEE) and DeepSDF [844] (©2019 IEEE) regress continuous implicit surface functions modelled by neural network.

### 14.1.2 Differentiable Rendering

Supervised learning on 3D data poses challenges due to the difficulty and expense of obtaining large-scale ground-truth annotations. To overcome these issues, a graphics rendering pipeline has been integrated into neural network training, leading to the advancement of Differentiable Rendering (DR). DR provides this capability by allowing gradients to flow from 2D images back into the 3D scene parameters (Figure 14.2). In a graphics pipeline, a 3D model may be represented in one of two ways:

- **Surface data**, which explicitly encodes object boundaries (*e.g.*, polygons or implicit surfaces).
- **Volume data**, which describes continuous fields such as density or color, representing an object’s internal properties and material distribution.

The rendering processes for these representations are called **surface rendering** and **volume rendering**, respectively. Accordingly, different rendering methods are used for each representation (Figure 14.3):

- Surface rendering often employs a method called **rasterization**, which projects surface geometry onto a 2D plane. It leverages hardware acceleration, making it highly efficient for explicit surfaces.
- Volume rendering is typically implemented via **ray marching**, which casts rays through a volume, sampling values along each ray’s path. While well-suited for continuous fields, it requires higher computational cost.

Although early DR approaches used softened rasterization for meshes [692, 540, 677], volume rendering has gradually adopted as a more powerful approach for gradient-based optimization. By providing gradients over 3D space —even before identifying exact surface positions— volume rendering makes neural network training easier and thus is widely adopted as a DR backbone. Following NeuralVolume [685], works like differentiable volumetric rendering [804] and Scene Representation Networks [1012] use differentiable ray marching to represent implicit surfaces. Although initially limited to simple shapes, these methods laid the groundwork for more complex and realistic 3D reconstructions.

### 14.1.3 3D Scene Representation with Differentiable Rendering

A major breakthrough occurred in 2020 when Mildenhall et al. [761] introduced NeRF, a technique for scene reconstruction that optimizes a MLP using volumetric ray-marching from posed color images. NeRF employs a single MLP as the scene representation, performing volume rendering by marching along pixel rays and querying the MLP for opacity and color. Given that volume rendering is naturally

*differentiable rendering*

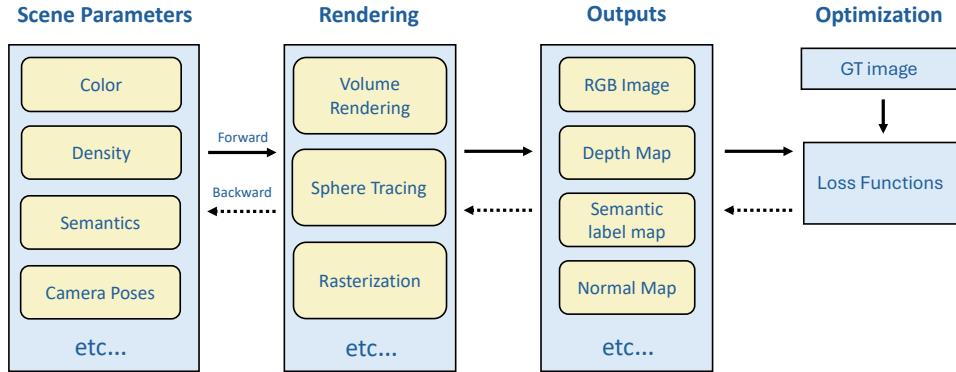


Figure 14.2 Overview of differentiable rendering pipeline. DR allows to optimize 3D scene parameters from multi-view 2D images in a end-to-end manner thanks to its consistent gradient propagation.

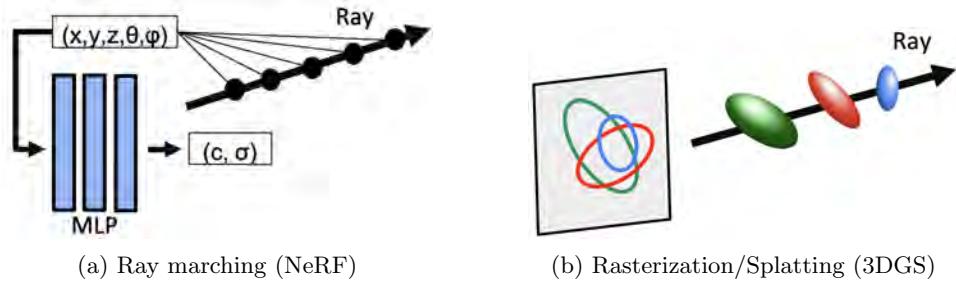


Figure 14.3 Comparison of the rendering methods in NeRF and 3D Gaussian Splatting. In ray-marching based approaches, the inference results of points sampled at fixed intervals along a ray are blended, whereas in rasterization, each primitive along the ray is projected onto the image plane and then blended.

differentiable, the MLP is optimized to minimize rendering loss using multiview information, achieving high-quality novel view synthesis. NeRF’s MLP inputs include spatial location and viewing direction, combined with positional encoding to capture high-frequency scene details. Despite achieving high-quality novel view synthesis, NeRF’s primary bottleneck is its training speed. To address this, recent developments have introduced explicit volume structures, such as voxel grids [1047, 341] or hash functions [781], to enhance performance. These advances have shown that the key to high-quality novel view synthesis lies in differentiable volumetric rendering rather than the neural network itself. They have demonstrated that it is possible to achieve rendering quality comparable to NeRF without relying on an MLP. However, even in these improved systems, per-pixel ray marching continues to be a significant bottleneck for rendering speed.

In parallel to the explosion of NeRF and volume rendering, differentiable point-based rendering techniques have also been researched [596, 951, 555]. These methods use point-based representation such as a point cloud [596, 951] or metaball [555], and define a differentiable rendering pipeline using a CNN based rasterizer or approximated renderer. In 2023, Zwicker et al. [1316] proposed a radically different point-based rendering approach for radiance field capture based on 3D Gaussian Splatting [551] (3DGS). 3DGS represents a scene using numerous semi-transparent Gaussian blobs that are rendered via alpha-blending. Unlike NeRF, 3DGS employs differentiable rasterization (Figure 14.3). Similar to traditional graphics rasterization, 3DGS iterates over the primitives to be rasterized rather than marching along rays. This method leverages the natural sparsity of a 3D scene, resulting in a representation that is expressive enough to capture high-fidelity 3D scenes while offering significantly faster rendering with similar training speed to the most efficient volumetric NeRF.

In the following sections, we explore the technical details of NeRF and 3DGS (Figure 14.4). Despite their shared goal of radiance field capture, these methods differ significantly in scene representation and rendering pipeline. We begin with an overview of NeRF, focusing on its ray-marching-based differentiable rendering formulation and its various extensions, which utilize diverse data structures and evolve into generalized field representations known as Neural Fields. Following this, we examine 3DGS and its rasterization-based rendering pipeline, highlighting the key differences from NeRF.



Figure 14.4 Novel View Synthesis Examples of MipNeRF360 [58] (left, ©2019 IEEE) and 3DGS [551] (right). Both methods can capture high fidelity radiance field from posed RGB images taken in real-world. The images are from [58, 551]

## 14.2 Neural Radiance Fields (NeRF)

In this section, we explain the technical details of Neural radiance field (NeRF), its *Neural Radiance Fields (NeRF)* derivative methods, and its application to visual SLAM. This technical background

provides the fundamental knowledge needed to understand the motivations and differences with respect to 3D Gaussian Splatting in the next section.

#### 14.2.1 Method Overview

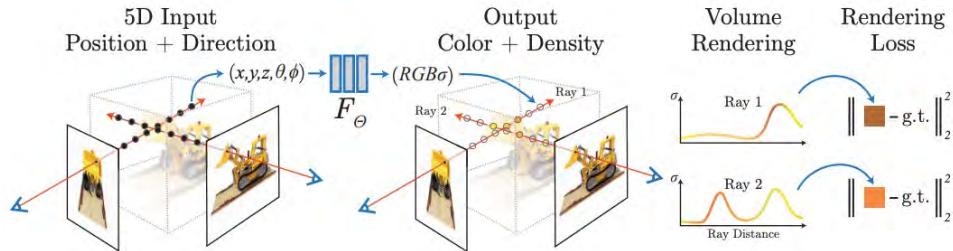


Figure 14.5 Overview of NeRF optimization pipeline. Radiance Field is represented by MLP and 2D images are rendered via differentiable volume rendering. MLP parameters are optimized to minimize the photometric error between rendered color image and ground truth color image. The image is from [761] (©2020 Springer).

The NeRF model is expressed as a function  $f_\Theta(\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ , with MLP weights  $\Theta$ , in-scene 3D coordinates  $\mathbf{x} = (x, y, z)$ , viewing direction  $\mathbf{d} = (\theta, \phi)$ , color  $\mathbf{c} = (r, g, b)$  and volume density  $\sigma$ . To synthesize novel views, the NeRF workflow involves casting camera rays through the scene, generating sampling points for each pixel. The local color and density at each sampling point are computed using the NeRF MLP(s), and volume rendering is used to synthesize the 2D image (Figure 14.5). The expected color  $C(\mathbf{r})$  of camera ray  $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$  is calculated using the following integral formulation:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt. \quad (14.1)$$

Here,  $dt$  denotes the differential distance traveled by the ray at each integration step and  $t_n$  and  $t_f$  denote near and far bounds. The terms  $\sigma(\mathbf{r}(t))$  and  $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$  represent the volume density and color at point  $\mathbf{r}(t)$  along the camera ray with viewing direction  $\mathbf{d}$ , respectively. The accumulated transmittance from  $t_n$  to  $t$  is given by  $T(t) = \exp\left(-\int_{t_n}^t \sigma(\mathbf{r}(s)), ds\right)$ . The continuous integral is estimated numerically with quadrature by dividing the ray into  $N$  evenly-spaced bins and sampling uniformly between bins and randomly within each bin. The volume rendering equation using the quadrature rule is formulated as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^N \alpha_i T_i \mathbf{c}_i, \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right). \quad (14.2)$$

$\delta_i = t_i - t_{i+1}$  denote the interval between consecutive samples  $t_i$  and  $t_{i+1}$ , while  $\sigma_i$  and  $\mathbf{c}_i$  indicate the density and color evaluated at sample point  $i$  along the ray, respectively. Additionally,  $\alpha_i = (1 - \exp(-\sigma_i \delta_i))$  denotes the opacity resulting from alpha compositing at sample point  $i$ .

To optimize the model, a quadratic photometric error between the rendered image and the ground truth color image is used:

$$L = \sum_{r \in R} \|\hat{\mathbf{C}}(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2^2, \quad (14.3)$$

where  $C_{gt}(r)$  denotes the ground truth color for the pixel associated with  $\mathbf{r}$  in the training image, and  $R$  denotes the batch of rays used for synthesizing the target image. In the context of SLAM, depth rendering through alpha-blending of opacity is often employed, to additionally minimize discrepancies between the expected depth and sensor or predicted depth.

#### 14.2.2 Data Structures in NeRF

The original NeRF paper [761] uses a single MLP to represent the entire radiance field and requires a forward inference to evaluate the color and density at specific 3D location. This means all the network parameters are involved with the evaluation and the forward inference and backward training is significantly slow. To achieve faster inference and training, several approaches introduced explicit data structure either partially or completely, and achieved fast data access.

**Voxel/Hash Grid.** Early works introduced a voxel grid to speed up the forward inference. Later on, Plenoxel [341] and DirectVoxGo [1047] introduced voxel feature-grid as trainable properties and encode radiance field into it. Compared to MLP based NeRF, color and density at specific location is queried by simple index lookup and trilinear interpolation, which makes the whole pipeline significantly faster (from days to 10mins). Instant-NGP [781](Figure 14.6) further introduced multi-resolution hash encoding, efficiently encodes volumetric features in a hash table, and together with highly optimized CUDA implementation, it achieves NeRF training in a few seconds.

**Points and Surfels.** Point-based representations have also been used in conjunction with NeR (Figure 14.7). Point-NeRF [1206] optimizes point-clouds with features and tiny MLP decoder, and render the image via a volume rendering framework. It demonstrated that the point-based representation has a flexibility of spatial allocation compared to regular voxel grids, which allows the model to skip empty spaces, resulting in a speed-up of a factor of 3 over vanilla NeRF. Similar to point clouds, surfels can also be used to represent radiance field [366].

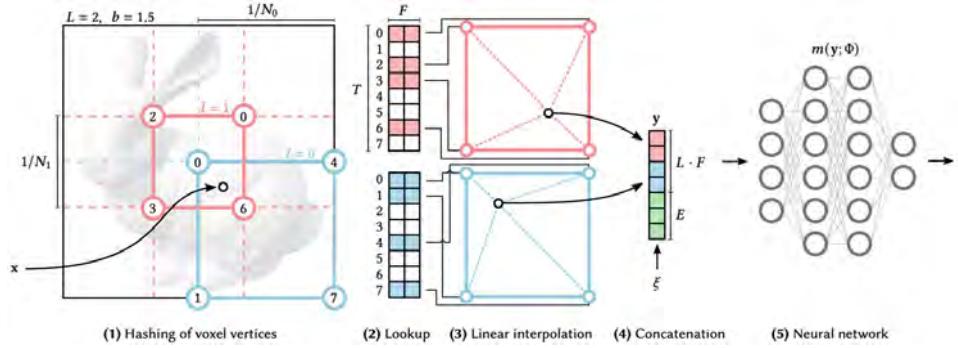


Figure 14.6 An example of grid-based NeRF (Instant-NGP [781]). The radiance field is modelled by a combination of multi-resolution hash grid and tiny MLP decoder. The image is from [781]

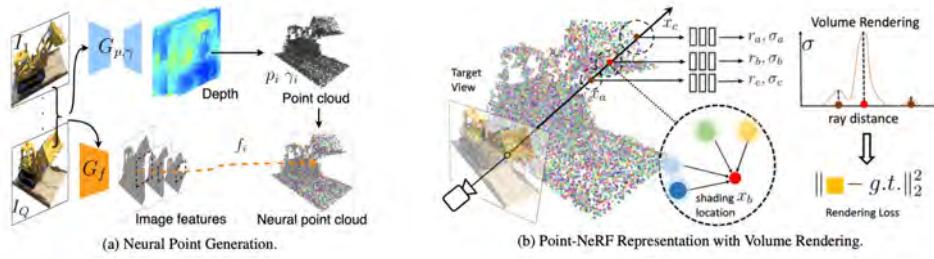


Figure 14.7 An example of point-based NeRF representation (Point-NeRF [1206]). The radiance field is modeled by a combination of points with trainable features and tiny MLP decoder. The image is from [1206] (©2022 IEEE).

### 14.2.3 Neural Fields

While initial NeRF work primarily focuses on representing a radiance field — mapping 3D positions to color and density for photo-realistic scene capture — the underlying concept can be extended to various other outputs, depending on the downstream task. Such generalized field representations are termed *Neural Fields*. Here, we introduce a few major examples.

**Surface Representations (SDF and Occupancy).** To achieve more geometrically accurate 3D reconstructions from RGB images, Neural Fields can utilize representations like Signed Distance Functions (SDF) and Occupancy Fields (Figure 14.8). Techniques such as NeuS [1156] and VolSDF [1238] leverage SDF to represent surfaces. SDF encodes the distance of a point from the nearest surface, with the sign indicating whether the point is inside or outside the object. By integrating SDF into NeRF, it is possible to achieve precise surface modeling. Techniques like UNISURF [816] employ occupancy fields to represent the presence or absence of

surfaces in the 3D space. Occupancy fields divide the space into a grid, where each cell indicates whether it is occupied by a surface. This method can be combined with NeRF to handle complex scenes effectively. To extract meshes from these representations, techniques such as marching cubes can be applied to convert the continuous surface representation into a discrete mesh format.

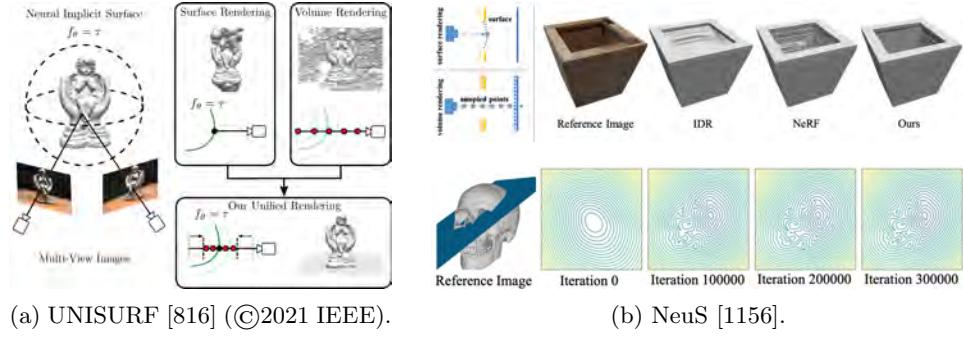


Figure 14.8 Neural Field for surface reconstruction. UNISURF represents occupancy field and NeuS represents continuous Signed Distance Functions via MLP which can be rendered via volume rendering. The images are from [816, 1156]

**Semantics.** Incorporating semantic information into Neural Fields enables the decomposition of scenes into meaningful components, enhancing scene understanding and interaction (Figure 14.9). Early works such as [1288, 1289] extend NeRF by outputting semantic labels in addition to color and density. This is achieved by training the model with multi-view semantic images, enabling it to predict the semantic category of each part of the scene. These fields fuse high-dimensional features from multiple views to produce a 3D consistent semantic feature field. More generally, NeRF can fuse 2D features from pre-trained foundation networks and perform more advanced semantic tasks such as scene decomposition and interactive editing via language prompt or clicking [585, 1108, 742].

#### 14.2.4 NeRF/Neural Fields for Visual SLAM

Neural Fields were initially designed for offline novel view synthesis using posed color images. However, their ability to produce high-fidelity reconstructions has driven interest in applying them to Visual SLAM, which requires the real-time joint estimation of camera pose and scene representation.

Neural Field-based 3D representations offer several advantages over classical dense SLAM methods. First, end-to-end optimization via differentiable rendering simplifies 2D-to-3D data fusion, eliminating the need for multiple hand-designed fusion steps as in TSDF-based SLAM, where consistency is not always guaranteed. Second, Neural Fields can model complex light fields, handling transparent objects

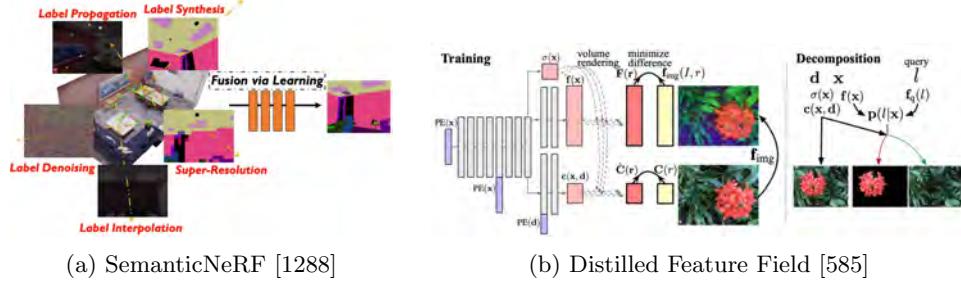


Figure 14.9 Neural Field for semantic scene reconstruction. By using additional semantic head for MLP, NeRF can output multi-view consistent semantic label/features which can be trained by per-view semantic prediction. The images are from [1288, 585]

and reflections more flexibly for photometric error minimization. Third, depending on the data structure, they enable scene completion and compression, leveraging priors inherent in the neural representation.

iMAP [1044] (Figure 14.10) was the first work in this area, utilizing a purely MLP-based map representation similar to the original NeRF. The key insight of iMAP is that an MLP-based approach benefits from map compression and scene completion due to the inherent smoothness prior in MLPs. Similar to the development of NeRF research, the integration of classical and neural representations has also been investigated in the SLAM context. NICE-SLAM [1306] was the first to introduce a hybrid representation combining a multi-resolution voxel feature grid with a tiny MLP. Similar grid-type representations have been adopted by other works [1229, 524, 1150]. NeRF-SLAM [937] accelerates NeRF reconstruction by leveraging depth information from DROID-SLAM (*cf.* Chapter 13). More recently, PointNeRF-style point primitives have also been applied to SLAM [965]. Among the various technical components, the choice of underlying data structure has the greatest influence on reconstruction quality, computational efficiency, and real-time performance in SLAM. Given this, we next examine the various data structures used in Neural-Field SLAM methods and detail their key characteristics and trade-offs.

#### 14.2.4.1 Data Structure Characteristics and Trade-Offs in Neural Field SLAM

The choice of underlying data structure significantly impacts the properties of Neural Field-based SLAM. Table 14.1 provides an overview of various Neural-Field SLAM methods. Below, we detail the major characteristics and trade-offs associated with these representations.

**Trade off Between Compression and Inference Speed.** While the initial Neural Field SLAM work uses a single-MLP, more recent methods use (partially or entirely) more explicit data structures similar to the classical graphic primitives. MLP approximates the entire radiance field function via a network’s parameters



Figure 14.10 Qualitative results of iMAP [1044]. The method firstly demonstrated that MLP and differentiable rendering framework can be used for real-time joint estimation of camera pose and scene geometry from RGB-D video stream.

	MLP	Voxel/Hash	Point/Surfel
Compression	Good	Moderate	Moderate
Inference speed	Limited	Good	Moderate
Scene Completion	Good	Good	Moderate
Dynamic Allocation	Limited	Limited	Good
Forgetting Problem	Limited	Moderate	Moderate

Table 14.1 *A comparison of underlying data structure for Neural Field.*

and has huge benefits in terms of compression, but the physical meaning of the parameters is hard to interpret. On the other hand, explicit data structures like voxel grids are more interpretable and allow fast data access and local update of the map, but they require more memory. To that end, the map representation exhibits a typical time-space complexity trade-off. For SLAM tasks in particular, real-time operation is paramount and therefore explicit data structures are often preferable in practice.

**Scene Completion.** iMAP demonstrated that MLP-based 3D reconstruction has hole-filling capability of the unobserved region, thanks to the continuous nature of the function. Multi-resolution grid also has similar property (Figure 14.11). This is because the positional encoding or multi-resolution grid resolution smooths the signal within the frequency or grid resolution. This is not observed by point-type representation where the primitive is discrete and does not have global smoothing.

**Dynamic Allocation of Graphic Primitives.** Unlike offline scene reconstruction tasks, SLAM requires the online update of state variables based on run-time observations to adjust for dynamic changes, such as scene boundaries and loop closures. In Neural Fields, MLP and voxel-type representations are difficult to adjust for these updates due to their fixed boundaries and positional encoding/voxel resolution, which requires dynamic allocation of multiple local NeRF models [1067]

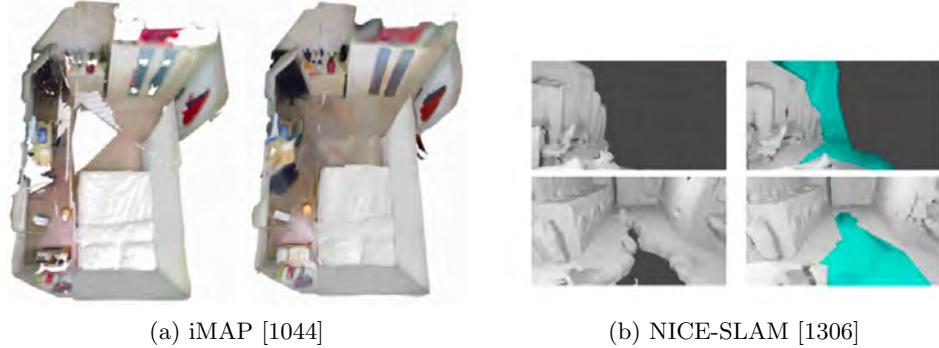


Figure 14.11 Hole-filling effect of NeRF-based SLAM. Thanks to the use of multi-resolution frequency or grid encoding and continuous MLP function, the representation has a capability to smooth local geometry and provides plausible hole-filling effect of small unobserved regions. The images are from [1044] and [1306] (©2022 IEEE).

(Figure 14.12). On the other hand, point representations offer more flexibility in primitive allocation and can handle loop closures more effectively [670].

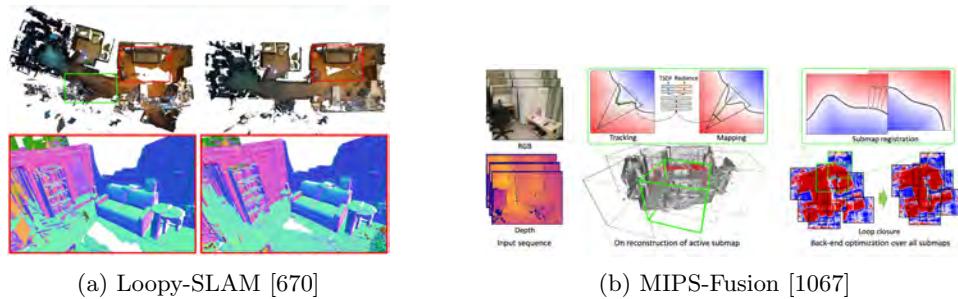


Figure 14.12 Loop Closure on NeRF-based SLAM methods. While voxel-based approaches require multiple local submap of Neural Field to handle camera pose update by loop closure, discretized point-based representation can more seamlessly change the primitive allocation. Images are from [670] (©2024 IEEE) and [1067] (©2023 ACM).

**Forgetting Problem.** In both purely MLP-based and hybrid representations, the existence of a global function implies that any local parameter update impacts the overall results. Consequently, the map must retain past information as a training signal within the optimization window and requires an explicit keyframe database. This issue, often referred to as the “forgetting problem,” can be mitigated by reducing reliance on or completely removing the global function, typically represented by an MLP.

### 14.3 3D Gaussian Splatting



Figure 14.13 Novel view rendering results of 3DGS. (Left) RGB rendering. (Right) Shaded Gaussians. 3DGS represents the scene by allocating a large number of semi-transparent 3D Gaussians.

While NeRF performs differentiable ray-marching, 3D Gaussian Splatting (3DGS) performs differentiable rasterization for volume data. Similar to regular graphics rasterization, by iterating over the primitives to be rasterized rather than marching along rays, 3DGS leverages the natural sparsity of a 3D scene and achieves a representation which is expressive enough to capture high-fidelity 3D scenes while offering significantly faster rendering (Figure 14.13). The input consists of a set of color images with corresponding poses estimated by SfM. In addition, 3DGS takes the sparse point cloud generated during the SfM process, which is used for constructing a set of initial 3D Gaussians as graphical primitives.

3D Gaussian Splatting  
(3DGS)

#### 14.3.1 Method Overview

Each 3D Gaussian is parameterized by its 3D mean position  $\mu \in \mathbb{R}^3$ , covariance matrix  $\Sigma \in \mathbb{R}^{3 \times 3}$ , opacity  $o$ , and direction-dependent color  $\mathbf{c}$  represented by spherical harmonics (SH). Given a scaling matrix  $\mathbf{S} \in \mathbb{R}^{3 \times 3}$  and rotation matrix  $\mathbf{R} \in \text{SO}(3)$ , the covariance matrix  $\Sigma$  is represented as :

$$\Sigma = \mathbf{R} \mathbf{S} \mathbf{S}^\top \mathbf{R}^\top. \quad (14.4)$$

These 3D Gaussians are projected onto the image plane ( $z = 1$  plane) via an approximated projection function for differentiable rasterization. The projected Gaus-

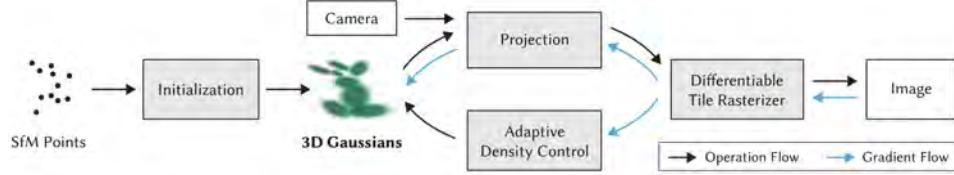


Figure 14.14 Overview diagram of the 3DGS optimization process [551].

sian's covariance matrix in 2D image space  $\Sigma'$  is formulated as:

$$\Sigma' = \mathbf{J}\mathbf{W}\Sigma\mathbf{W}^\top\mathbf{J}^\top, \quad (14.5)$$

where  $\mathbf{J}$  denote Jacobian of the projective transformation and  $\mathbf{W}$  is viewing transformation matrix. The upper left  $2 \times 2$  block of  $\Sigma'$  is used as a 2D covariance matrix. We obtain the value of the 2D Gaussian function by querying a pixel  $\mathbf{x}_{2D} = [u, v]^\top$ .

The final color  $C$  of the pixel  $\mathbf{x}_{2D}$  is determined through alpha blending of the 3D Gaussians:

$$C = \sum_{i \in \mathcal{N}} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (14.6)$$

where  $\mathcal{N}$  is the set of Gaussians along the pixel's ray, and  $\alpha_i$  is the product of the opacity  $o_i$  the pixel-space Gaussian function:

$$\alpha_i = o_i G_{2D}(\mathbf{x}_{2D}). \quad (14.7)$$

In alpha-blending, the 3D Gaussians are sorted in depth order relative to the rendering viewpoint. This sorting process is performed for each tile of the input image, which is divided into  $16 \times 16$  tiles, using a GPU-optimized radix sort. Moreover, for efficiency, only the Gaussians that fall within the current view frustum are considered (Frustum Culling). The parameters of the 3D Gaussians are estimated by minimizing a weighted sum of the L1 loss and the SSIM loss between the rendered image and the ground truth image, with a weight  $\lambda$

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{SSIM}. \quad (14.8)$$

Furthermore, during the optimization process, adaptive densification and pruning are performed to flexibly adjust the number of Gaussians based on their gradients and opacities (Figure 14.14).

#### 14.3.2 Applications of 3D Gaussian Splatting

Similar to NeRF's evolution into a generic Neural Field, 3DGS has been applied to various 3D vision tasks. Given the large volume of research, we highlight a few early papers in this area.

**Surface Reconstruction.** 3DGS is primarily designed for photorealistic novel view synthesis and does not inherently ensure geometric accuracy. Several works address this limitation (Figure 14.15). SuGAR [411] introduces geometric regularization to 3DGS, enabling geometrically accurate 3D mesh extraction. It defines surface normals for each 3D Gaussian and enforces local surface smoothness by encouraging neighboring Gaussians to be as flat as possible. 2D Gaussian Splatting (2DGS) [483] replaces 3D Gaussians with 2D disks, explicitly defining surface normals within the graphic primitives. By incorporating normal and depth consistency losses, 2DGS further improves surface reconstruction accuracy.

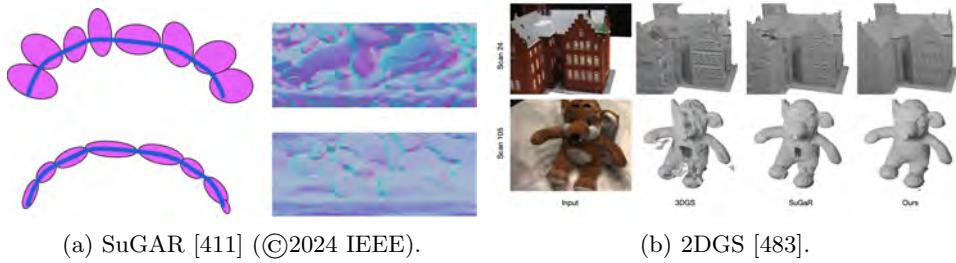


Figure 14.15 Examples of surface-oriented Gaussian Splatting methods. By defining surface normal directions and enforcing smoothness between neighboring Gaussians, these methods achieve accurate surface reconstruction. Images are from [411] (©2024 IEEE) and [483].

**Semantic Scene Understanding.** 3DGS can also be extended to semantic scene understanding. Like NeRF, the most common approach propagates 2D semantic segmentation to 3D Gaussians with semantic features via end-to-end training enabled by differentiable rendering. For instance, LangSplat [892] and FMGS [1313] integrate 2D foundation models such as CLIP and DINO into multi-view consistent 3D Gaussians during the 3DGS training process, enabling open-vocabulary semantic segmentation (Figure 14.16). Since 3D Gaussians are explicit discrete representations, editing the reconstructed scene (*e.g.*, manipulation or removal) is more straightforward than with the continuous NeRF representation.

#### 14.3.3 3D Gaussian Splatting for SLAM

3DGS has achieved remarkable success in real-time novel view synthesis, and its potential for SLAM is currently under active exploration. Similar to NeRF, 3DGS leverages end-to-end optimization via differentiable rendering to seamlessly unify localization and mapping. This integration eliminates the need for multi-stage, hand-designed algorithms and enables the modeling of complex light fields —including those produced by transparent objects— that are challenging to capture with classical 3D representations.

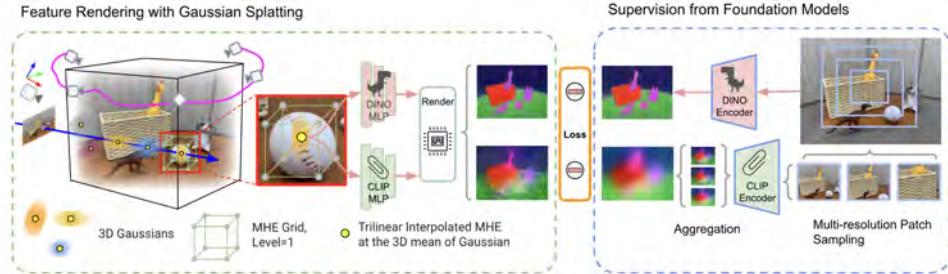


Figure 14.16 An example of Gaussian Splatting method for semantic scene reconstruction. Multi-view 2D image features are fused into 3D Gaussians via differentiable rendering. The image is from [1313] (©2024 springer).

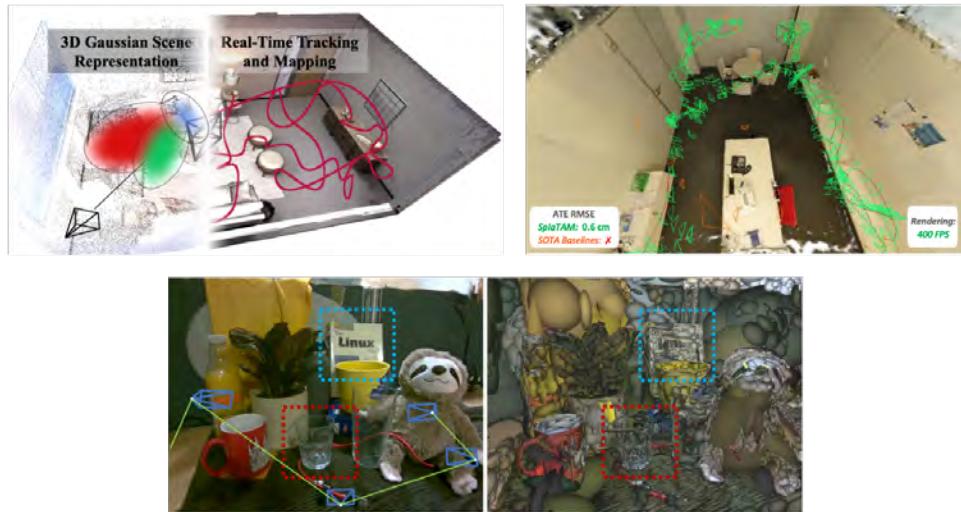


Figure 14.17 Examples of 3D Gaussian Splatting-based visual SLAM methods [1211, 547, 737]. Images from [1211] (©2024 IEEE) and [547] (©2024 IEEE).

Beyond these similarities, 3DGS offers several unique advantages over NeRF. First, its efficient, rasterization-based pipeline enables significantly faster image rendering, directly benefiting downstream applications that require real-time map interaction (*e.g.*, robot navigation). Second, its representation as discrete, point-like Gaussian primitives allows for flexible allocation and updates of scene elements. This flexibility is particularly well-suited for SLAM tasks where state estimates must dynamically adapt to runtime observations, such as scene boundary updates and loop closures.

Leveraging these properties, 3DGS can serve as a core scene representation in visual SLAM systems (Figure 14.17). Several early works have explored its inherent

capabilities for simultaneous localization and mapping [1211, 547, 737]. A straightforward application is RGB-D SLAM, where an external structured sensor—such as a Time-of-Flight (ToF) depth camera—provides accurate, dense depth data that can be used as a prior or ground-truth signal for the positions of Gaussians. By incorporating this depth information, the positions of the 3D Gaussians can be accurately initialized and further regularized by minimizing a depth rendering loss, leading to accurate camera tracking as well as high-fidelity and fast map rendering.

However, these approaches require additional hardware to reconstruct 3D Gaussians, even though the original 3DGS method only needs a commodity camera to capture RGB images. Ideally, the method should operate solely in a vision-based monocular SLAM setting by fully leveraging the flexible and optimizable nature of 3D Gaussians. In this context, work such as MonoGS [737] has demonstrated that purely color image-based monocular SLAM is indeed possible (Figure 14.18). In MonoGS, even though the initial locations of the 3D Gaussians are unknown, they can be incrementally estimated by randomly initializing the Gaussians and refining their positions through multi-view optimization.

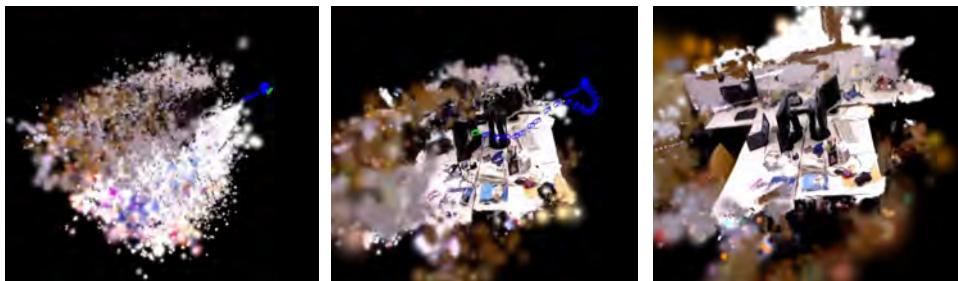


Figure 14.18 Monocular Reconstruction result of MonoGS [737]. Thanks to 3D Gaussian’s flexibility of its spatial allocation, the method can perform joint estimation of camera pose and scene reconstruction only from color images without any depth supervision.

The discrete nature of 3D Gaussians provides significant advantages for global map updates, particularly in the context of loop closure. Because these representations consist of a collection of individual, well-defined primitives, it becomes easier to match and align various segments of a scene, even as the environment evolves over time. For instance, LoopSplat [1305] demonstrates a practical application of this concept. In their approach, the discrete 3D Gaussian elements serve as robust features that can be reliably registered across different time instances and viewpoints. Another notable advantage of 3DGS as a SLAM representation is its ability to capture complex light fields. By effectively capturing the intricate distribution of light within an environment, 3DGS not only enhances visual fidelity but also provides additional cues that can improve both mapping and localization performance.  $I^2$ -SLAM [42] fully exploits this property by integrating a more precise physics-based

rendering process, such as motion blur and tone mapping, into the existing 3DGS framework. This integration enables simultaneous HDR map reconstruction and accurate camera pose tracking (Figure 14.19).

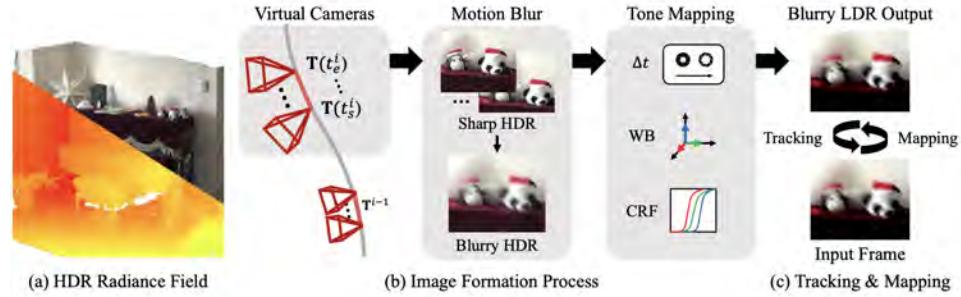


Figure 14.19 The system over view of  $I^2$ -SLAM [42] (©2024 Springer). The method integrates physical imaging process into SLAM to capture HDR radiance field.

While the full potential of 3D Gaussian Splatting (3DGS) as a unified SLAM representation is still under active investigation, another promising research direction involves integrating 3DGS into existing point-cloud processing methods. Since 3DGS can be treated as a point cloud by simply omitting its additional features, it becomes straightforward to combine with well-established point-based algorithms that are known for their stability and practical performance in real-world applications (Figure 14.20). For example, Photo-SLAM [488] combines a sparse tracking system (ORB-SLAM) with 3DGS, while both GS-ICP SLAM [424] and RTG-SLAM [865] leverage ICP for tracking. In the case of RTG-SLAM, the system further incorporates ORB keypoint-based backend optimization, resulting in fast and accurate performance with high practical applicability.

Furthermore, this inherent versatility of 3DGS extends to its fusion with LiDAR measurements. Several recent approaches have proposed hybrid LiDAR+3DGS frameworks [471, 1051, 622], demonstrating that 3DGS can serve as a unifying map representation across multiple sensor modalities. This wide range of applications, from RGB-based SLAM to LiDAR-enhanced reconstructions, demonstrates key advantages of 3DGS: its flexibility and adaptability, which make it an attractive choice for diverse SLAM scenarios.

#### 14.4 Further Readings & Recent Trends

NeRF/3DGS-based SLAM offers several advantages, including end-to-end optimization, complex light-field capture, real-time rendering, and dynamic primitive allocation, making it a promising research direction. However, significant technical

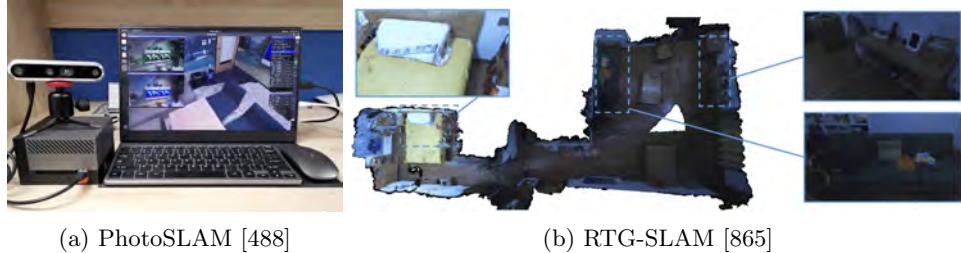


Figure 14.20 Example of the methods which combine 3DGS with keypoint-based SLAM. Images from [488] (©2024 IEEE) and [865] (©2024 ACM).

challenges remain. This section discusses these limitations and outlines new trends and potential future research directions.

**Real-time Performance.** One major challenge of NeRF/3DGS-based end-to-end SLAM is its extremely slow processing speed. While classical dense SLAM methods like ElasticFusion [1181] achieved real-time performance including tracking, mapping, and rendering, on a GTX 780Ti in 2015, most NeRF/3DGS-based SLAM methods struggle to exceed 5 fps even on modern GPUs with significantly higher FLOPs and VRAM (*e.g.*, NVIDIA RTX 4090). This limitation makes current NeRF/3DGS-based SLAM impractical for real-time robotics applications, restricting its use to offline server GPUs. Since 3DGS enables real-time rendering, optimizing it efficiently remains a key challenge. One potential direction for improving efficiency is adopting a compact representation that reduces the number of Gaussians, thereby lowering computational overhead. Another approach is employing a higher-order optimizer instead of standard first-order gradient descent to accelerate convergence. Alternatively, a learned network could predict both camera poses and 3D Gaussians in a single forward pass, eliminating the need for iterative optimization. In support of this approach, PixelSplat [169] and MVSplat [191] have demonstrated that a *feed-forward network* can predict 3D Gaussian parameters from posed images without any test-time optimization. Although these early works required camera poses estimated from an external module, pose-free 3D reconstruction methods based on feed-forward networks have begun to gain traction following the success of DUSt3R [1161] (see Chapter 13 for more details), which robustly predicts pointmap without relying on pose input. Building on this progress, NoPoSplat [1239] (Figure 14.21) has extended the concept to achieve pose-free 3DGS prediction, showing promising results. Exploring these techniques for multi-view, incremental SLAM settings is an interesting future direction.

**Camera Tracking Accuracy.** Camera tracking accuracy in NeRF/3DGS-based SLAM often falls behind classical sparse SLAM methods. These approaches rely on photometric consistency across multiple frames, performing well on synthetic datasets like Replica [1037], where ground-truth depth and color images are avail-



Figure 14.21 System overview of NoPoSplat [1239]. The method directly predicts 3D Gaussians from 2 input views, which can be used for pose estimation and novel view synthesis. The image is from [1239].

able. However, in real-world scenarios, this assumption often breaks down due to unmodeled effects such as sensor noise and rolling shutter. Improving accuracy may require more precise modeling of the scene’s light field and rendering process or exploring alternative matching paradigms beyond the color image domain. This challenge might also be addressed by incorporating a more robust feed-forward prediction model, as mentioned in the previous paragraph.

**4D Scene Reconstruction.** Currently, most NeRF/3DGS SLAM methods assume a static environment, while real-world scenes often exhibit significant dynamic changes. One common approach involves masking out potentially dynamic objects to focus solely on static background reconstruction for robust odometry estimation. However, considering the notable success of 3DGS in non-rigid/4D scene reconstruction and point-tracking, exploring a comprehensive 4D-SLAM framework appears to be a promising direction [138, 983, 1286, 738].

**Large-scale/Outdoor Scene Reconstruction.** Most NeRF/3DGS-based SLAM methods are currently limited to room-scale sequences, making large-scale and outdoor scene reconstruction a major challenge. Scaling up requires efficient map updates, such as adaptive resolution to manage computational load, and drift correction to maintain accuracy over extended areas. Outdoor environments introduce additional complexities, including illumination changes due to varying weather and time of day, which complicate light field modeling.

# 15

## Dynamic and Deformable SLAM

Lukas Schmid, Jose Maria Martinez Montiel, Shoudong Huang,  
Daniel Cremers, Jose Neira, and Javier Civera

Historically, SLAM systems have always faced challenges in environments where not only the sensor but also the scene is in motion. To address this challenge, data association—combined with approaches like RANSAC for outlier rejection—tries to distinguish which sensor data correspond to entities that present a coherent change of pose with respect to the sensor motion, from data picturing entities exhibiting independent motion. This is much in the same way that stars apparently move in unison in the sky, while planets wander, thus leading to their name.<sup>1</sup>

The methods described so far in this handbook have addressed the problem of estimating the robot pose and a representation of the environment based on sensor observations. However, an important assumption was that the environment is *static*, meaning that the scene remains unchanged and only the robot is moving. This assumption has several important implications. For example, Chapter 1 showed that if detected features remain in the same place, re-detecting them can directly be used to better constrain the robot pose. Conversely, Chapter 5 demonstrated how dense scene models can efficiently be reconstructed by *fusing* several posed observations if the underlying environment remains the same.

Nevertheless, real-world scenarios are often highly *dynamic*, thus violating the static assumption, where everything in the environment is rigidly attached and unmovable. For example, imagine what information an autonomous vehicle requires to capture in its map representation to drive safely through a city, where the environment includes not only a static background but also a variety of moving elements such as other vehicles, cyclists, and pedestrians. Or consider a home or service robot, mapping an environment where humans are frequently moving around the robot and impart changes on the scene, such as modifying objects or re-arranging a room. Or the even more extreme case of a surgical robot navigating through the human body with no static background at all where the entire scene is constantly deforming. Such factors make estimating both the robot state and a representation of the environment challenging problems in *dynamic* and *deformable* scenes.

To address these limitations, this chapter analyses the challenges of *dynamic* and

<sup>1</sup> The Greek term  $\pi\lambda\alpha\nu\eta\tau\eta\varsigma$  (planētēs) means wanderer.

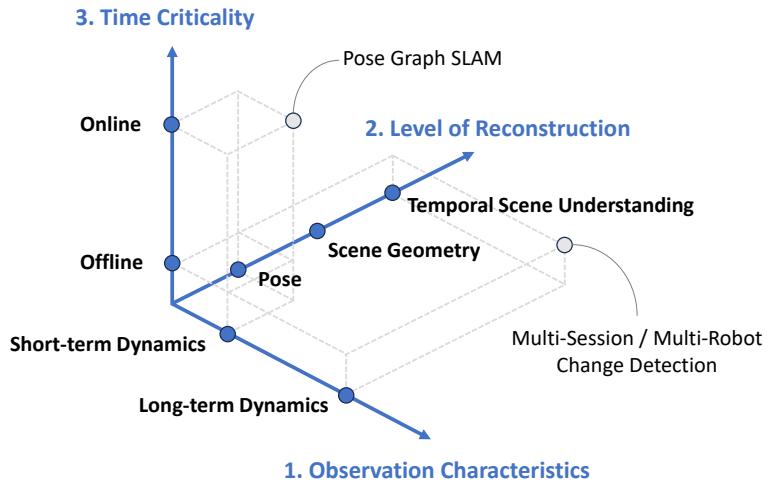


Figure 15.1 Overview of the main considerations for dynamic SLAM: 1) which dynamic effects are present in the observation of the robot, 2) at which level of detail is the robot required to understand the environment to accomplish its tasks, and 3) when and how fast does the robot need to process this information. For example, typical pose-graph SLAM can estimate the robot pose in real-time by rejecting dynamic parts as outliers. On the other hand, multi-session change detection aims to build a detailed understanding of the evolution of a scene from infrequent observations (sessions). Note that while some solution approaches can be well characterized in the space spanned by these axes, others may fill in an ‘area’ or ‘volume’ by addressing several aspects, such as both short and long-term dynamics, simultaneous state-estimation and scene representation, some components running online and some offline, as well as combinations across axes.

*deformable SLAM*. We first define and categorize different kinds of dynamic effects and objectives of dynamic SLAM in Section 15.1. We then introduce prevalent problems and current solution approaches for short and long-term dynamic SLAM in Section 15.2 and Section 15.3, respectively. We address the deformable SLAM case in Section 15.4. Finally, Section 15.5 revisits remaining challenges and outlines directions for future research in the rapidly expanding field of dynamic and deformable SLAM.

### 15.1 Characterizing the Dynamic SLAM Problem

Central considerations when modeling dynamic environments are (1) which dynamic effects are present in the observations of the robot, (2) at which level of detail the robot is required to understand the environment to accomplish its tasks, and (3) when and how fast the robot needs to process this information. While, ideally, a generalist robot will be able to build a rich description of any dynamic scene in real-time, oftentimes, it may be sufficient and more efficient to address

only the subset of problems that is relevant to a given application. An overview of these dimensions is shown in Figure 15.1, and each axis is further discussed below.

### 15.1.1 Characterizing Dynamics

Our approach to characterizing dynamics is based on how they manifest in the *observations of the robot*. We follow the definition of [690] and distinguish between *short-term* and *long-term* dynamics, referring to more *transient* or more *abrupt* characteristics in the *observations* of the robot, respectively. The key idea is that, fundamentally, almost all physical processes are continuous at short enough time scales. How different the observations of the robot are, and therefore whether the change characteristics are those of short or long-term dynamics, thus does not depend on the absolute duration of a phenomenon, but on the *relation* between the rate of change in the scene and the rate of observation by the robot. This relation is demonstrated in Figure 15.2, including several examples to illustrate how this applies to a broad range of time scales.

While the above definition is general and applies to vast time scales, it is important to point out that in most robotic applications, the relevant rate of change (humans or objects move typically at speeds  $\sim 10^{-1}$  to  $10^2$ m/s) is of the same order of magnitude as the rate of observation (most robotics sensors have a rate of  $\sim 10^0$  to  $10^3$ Hz). In this case, the distinction often simplifies to whether the robot is *currently observing* the motion or not. Then, *short-term* dynamics refer to all motion that is happening *within view* of the robot, where continuous observation of dynamic entities will result in *transient* change characteristics. For example, consider the smooth trajectory of a continuously observed rolling ball or the coherent motion of a human walking in front of the camera. We will discuss techniques to address *short-term* dynamics in Section 15.2.

On the other hand, *long-term* dynamics refer to dynamics happening *outside* the view of the robot. While the motion is still continuous locally, the robot will only observe the result of this motion that accumulated in between measurements, which is therefore typically of a more *abrupt* nature. For example, consider the case where a person picks up a bottle and places it again next to its initial position. If the robot only observes the table with the bottle before and after the person interacts with it, although the true motion was continuous, from the perspective of the robot, it will look like the bottle teleported between observations. An extreme example of this is the case of multi-session or multi-robot mapping, where the scene may be static during individual visits, but changes tend to accumulate between visits when the robot is not observing the scene.

As a result, an important implication of this view is that the difference between short and long-term dynamics depends on the *observation* of the robot and is not an inherent property of the environment or moving object, illustrated in Figure 15.3. However, the probability of long-term dynamics occurring is oftentimes corre-

*short-term dynamics*

*long-term dynamics*

*multi-session mapping*

*multi-robot mapping*

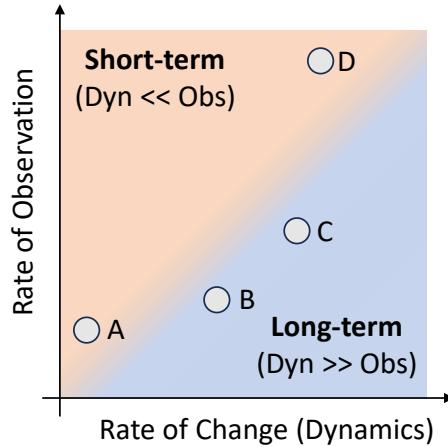


Figure 15.2 Definition of short vs. long-term dynamics: the dynamic characteristics apparent to a robot are given by the relation between the rate of change in the scene and the rate of observation by the robot, not by absolute time spans. Note that in most robotics domains, where the sensing rate is sufficient to capture continuous motion when in view, this simplifies to objects moving *within view* of the robot (short-term) vs. changing *outside* the view of the robot (long-term). Examples: A) Daily measurements of plant growth. Although the rate of observation is very low, the plants also only grow a small amount. As a result, appropriate methodologies to track the plants are similar to those for tracking people [175]. B) Daily patrolling of a building. Although the rate of observation is comparable, parts of the scene will have changed completely and long-term methods such as multi-session change detection are appropriate [564]. C) Object rearrangement behind the back of the robot. Although the robot may only be gone for a minute, the object configuration may have changed substantially between observations [973]. D) The Hawk-Eye System can track Tennis balls moving at up to 250 km/h by observing them at 340 Hz, leading to short-term dynamic observations and millimeter tracking accuracy [672].

lated with the time between observations. We will overview methods for *long-term* dynamics in section 15.3.

Finally, the cases discussed above typically assume that there is some *static background*, with various dynamic entities in the scene. However, in some relevant use cases like endoscopic surgery and medical robotics, *all of the scene* is deforming and moving around the robot, typically within the category described above as *short-term* dynamics. This setting where there is *no static* part of the scene is commonly referred to as *deformable SLAM*, which we discuss in Section 15.4.

#### 15.1.1.1 More Terminology

It is important to point out that in the community, a variety of further terms are used to describe related concepts to the ones introduced above. Instead of *short-*

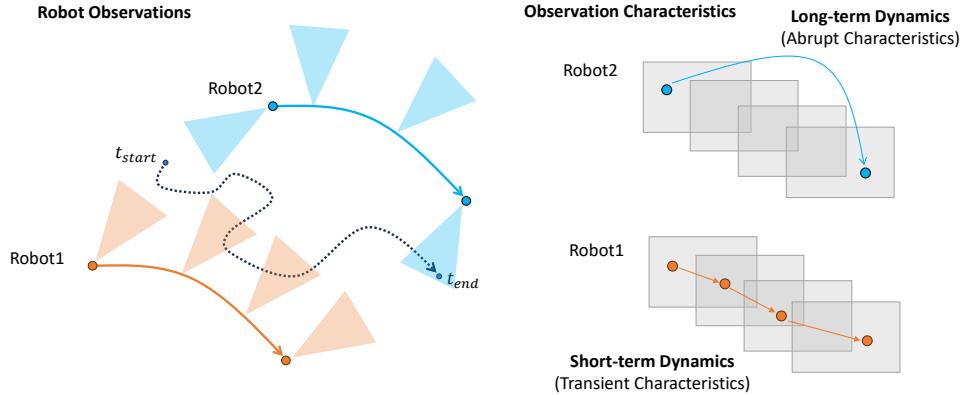


Figure 15.3 Role of the robot as observer. The same physical motion of an object (dotted line) can reflect as *short-term* dynamic for Robot1 and *long-term* dynamic for Robot2. Similarly, multi-session or multi-robot mapping can include cases where objects are moving while being observed (short-term dynamics), or changed outside of the view of the robot, most typically in between sessions or visits (long-term dynamics).

*term* dynamics, used terms include *dynamic* and *high-dynamic*, typically referring to objects that are currently moving (potentially within view of the robot).

On the other hand, to refer to *long-term* dynamic effects, terms including *semi-static*, *quasi-static*, or *low-dynamic* are used. These typically refer to objects that could move but are static while being observed, or changes that occur between multiple visits, which is a special case of long-term dynamics.

Beyond dealing with dynamic effects in 3D SLAM, certain methods aim to reconstruct a history, evolution, or higher-level understanding of the scene and its dynamics. This is often referred to as *spatio-temporal* or *4D* (being 3D space + time)<sup>2</sup> scene models.

In addition, the terms *lifelong*, *persistent*, and *continuous* are sometimes used to refer to SLAM, mapping, or localization systems that can address variable environments, often through operation over long time spans. Finally, note that not only geometric dynamics affect sensor readings, but also texture, illumination, and other changes (such as day/night or seasonal changes), in particular when using visual sensors [1166, 1005].

#### 15.1.1.2 Degrees of Dynamism

While the above characterization may seem to imply a clear classification into three cases—namely static background, short-term dynamics, and long-term dynamics—it is important to note that in practice the boundaries can be fuzzy, numerous different effects can combine, and each effect can be present in different intensities. For

<sup>2</sup> Note that this is different from the 4D or 3+1D used in radar SLAM (Chapter 9), where the fourth dimension refers to velocity measurements by the radar.

example, a scene may contain a single rigid moving object (*e.g.*, a car or box [688]), a single moving and deforming object (*e.g.*, a person [449]), only long-term changes (*e.g.*, regular patrolling of a building [321]), up to many moving and changing entities at the same time (*e.g.*, robots with humans in a home [973]), or even no static entities at all (*e.g.*, endoscopic surgery [890]). While, ideally, a generalist robot will be able to estimate all of these effects jointly, the problem becomes exponentially more complex the more effects are considered. Thus, prior knowledge about which dynamic effects can be expected or are relevant for the task at hand can enable the design of more specialized and thus better performing and more tractable solutions for a given scenario.

### 15.1.2 State Estimation vs. Scene Representation

As introduced in Chapter 5, SLAM is the dual process of estimating the *robot pose* as well as a *representation* of the environment. The same considerations apply to SLAM in dynamic environments. If only estimating the robot state is sufficient for an application, it is oftentimes adequate to treat dynamic observations as outliers or noise. In this case, similar to SLAM in static scenes, accurate results can be achieved by ignoring or rejecting the corresponding features or measurements and tracking or localizing with respect to the static background. We will discuss considerations to this effect, tailored to short-term and long-term dynamics, in Section 15.2.1 and Section 15.3.1, respectively. A special case is the setting of deformable SLAM, where there are *no static* parts to localize against. We will address this setting in Section 15.4.

Conversely, if a dense map is desired, reconstructing a moving and changing scene requires that the robot can detect and represent some or all motion and changes in the scene. We will discuss representations and techniques for short-term dynamics in Section 15.2.3, long-term dynamics in Section 15.3.2, and first unified dynamic SLAM methods in Section 15.3.3. Beyond reconstruction, we will briefly introduce methods for higher-level understanding of dynamic and changing scenes in Section 15.3.4.

### 15.1.3 Online vs. Offline Methods

*online vs. offline*

The final axis we consider is the time criticality of a perception system, meaning whether dynamics must be handled in real-time or whether it suffices to process the data offline. Since most robotic applications require that robots interact with their environment immediately, especially if the scene is dynamic, this chapter primarily focuses on real-time methods. However, in certain applications such as regular monitoring of a building, it may suffice to collect the sensor data and then process it offline. Similar to (offline) Structure from Motion vs. (online) SLAM, this has the advantage of additional computational resources, which facilitates processing at

higher resolutions and generally can achieve more accurate results. An additional notable difference to online methods is the fact that all data is already available. Specifically, to detect moving or changed objects in frame  $F_t$  at time  $t$ , all future measurements  $F_{t+1}, \dots, F_T$  are already available which can provide essential information not available to online methods. While this chapter focuses primarily on online methods, we briefly introduce the relevant offline problems of map cleaning and change detection in Section 15.3.2 and NRSfM in Section 15.4.1.

Finally, while Part II of this handbook was organized by sensing modality, the implications of dynamic scenes for SLAM are often similar across many sensing modalities. We therefore structure this chapter according to different dynamics characteristics. Nonetheless, each section will discuss considerations for different sensing modalities, where so far predominantly visual, LiDAR, and proprioceptive sensors have found notable use for dynamic and deformable SLAM.

## 15.2 Short-term Dynamics and Dynamic SLAM

In the following sections we further discuss the implications of individual short-term dynamic objects for SLAM. Historically, first approaches used data association to identify measurements that can be explained by the sensor moving, and distinguish them from the ones corresponding to objects independently changing their pose. Initial techniques included statistical tests [794], as well as classical algorithms such as RANSAC [222] and the Hough transform [1070]. These methods allowed simply discarding measurements of dynamic objects as outliers. Later, efforts turned to not simply ignoring dynamic objects, but also trying to reconstruct the environment regions occluded by them, either by interpolating missing data [1068], recovering data from alternative sensor poses where no occlusion occurred [75], or by image ‘inpainting’ using machine learning [77]. The result is much more useful for localization because there are no ‘holes’ in the rendered map. We will address dynamic objects as outliers in Section 15.2.1.

*data association*

An alternative also pursued in related work is to track the dynamic objects as part of the SLAM framework. In computer vision, this problem is known as *multi-body structure from motion*, an extension to classical SfM. The mathematical foundations for the solution, as well as practical algorithms have much in common with the SLAM perspective [832]. This was initially deemed unfeasible due to high computational cost [1146]. Recently, it has become feasible [76], and it might be necessary in order for the robot to avoid collisions with dynamic objects. We discuss dynamic object tracking in Section 15.2.2.

To provide an even more detailed understanding of dynamic scenes, one can further aim to reconstruct the scene in a dense manner, including the dynamic objects. Having access to detailed moving object models can be useful for applications such as manipulation, where the robot needs to understand the object shape and motion in order to interact with it. We discuss dense dynamic SLAM in Section 15.2.3.

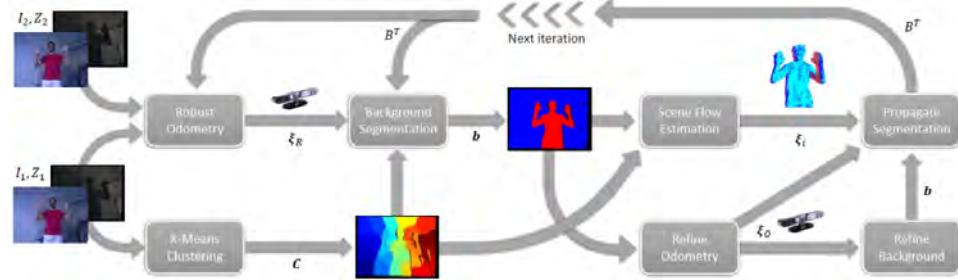


Figure 15.4 In [509], a method is introduced that jointly estimates the camera motion and the 3D scene flow for moving objects from an RGB-D video sequence. Initial robust odometry is used to segment the clustered frame into dynamic and static (background) based on the residuals of each cluster. The segmentation can then be used to further refine the odometry. (©2017 IEEE)

*rigid scene model*

*inertial measurement unit  
leg odometry*

### 15.2.1 Dynamic Object Removal

One of the simplest, but still effective approaches to SLAM in partially dynamic scenes consists in discarding sensor measurements of the dynamic parts of the scene and removing such parts from the map states. The rationale is twofold. First, persistent objects and landmarks are oftentimes the main focus of mapping, dynamic entities being in many occasions of limited interest (such as pedestrians passing by). Second, scene rigidity is assumed by most geometric estimation techniques, and hence dynamic parts can be detected and classified as spurious measurements to rigid models. To this end, a large variety of approaches has been proposed. We summarize the main families of methods below. Note that modern dynamic SLAM methods oftentimes employ combinations of these principles for best performance.

**Proprioceptive Sensing.** An essential property of proprioceptive sensing is that, by definition, it does not rely on observations external to the robot. For this reason, it can also not be influenced by external effects, such as motion in the scene. Since inertial and internal odometric measurements carry information of the robot states they help disambiguating the motion corresponding to the robot and that corresponding to dynamic parts of the scene from relative measurements [1021]. We refer the reader to Chapter 11 and Chapter 12 for details on how to fuse inertial or kinematic measurements with exteroceptive ones, respectively.

**Robust Estimation.** Most scenes and most sensors provide a set of measurements that is significantly larger than the minimal set for geometric estimation. In these cases, and for low rates of dynamic content, redundancy can be used to identify measurements of dynamic objects as outliers to rigid geometric models. Two methods are mainly used for that. First, RANSAC [328] is commonly used to initialize the state estimates by selecting a minimal sample set with the largest consensus among the whole measurement set. However, this may fail in cases of

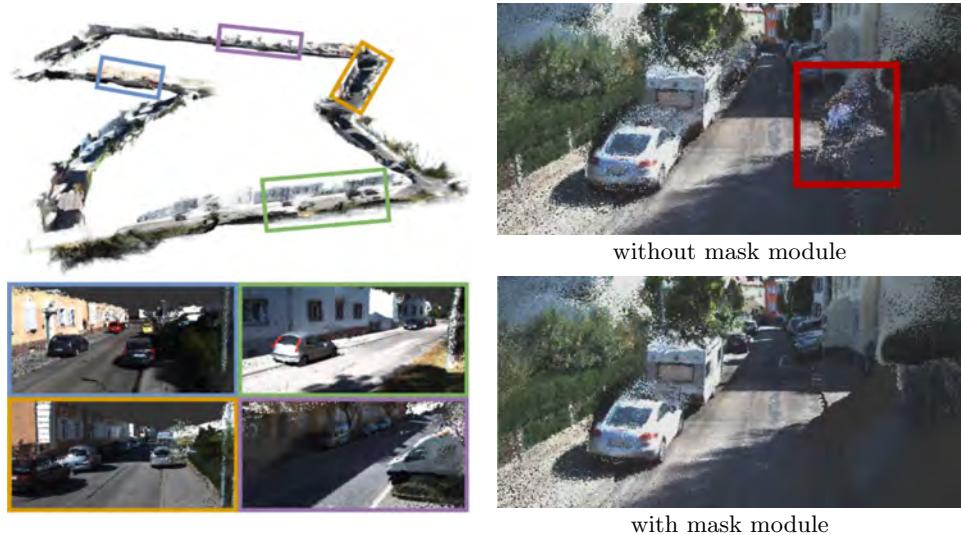


Figure 15.5 In MonoRec [1184], a deep network is trained to recover a dense reconstruction from a single moving camera. Moving objects are filtered out by a *mask module* that taps into a brightness consistency cost volume to identify structures (colors across subsequent frames) that are not compatible with the dominant ego motion. (©2021 IEEE)

large occlusions where the largest consensus set may be that of a moving object. Second, after initialization and during iterative optimization, an alternative to the typical  $\ell_2$  norm is to use robust loss functions. These have sub-quadratic growth for large residuals, which are assumed to be outliers, and hence their influence on the states is reduced. For further details on RANSAC and robust losses, we refer the reader to Chapter 3.

*robust loss function*

**Multi-View Motion Cues.** In addition to discarding dynamic content at initialization and reducing or even canceling its influence by robust loss functions, another common approach is removing dynamic states once there is sufficient evidence of non-rigidity from its measurements. Evidence is typically implemented as a threshold on the accumulated number of high-residual measurements. Figure 15.4 shows an example of this where RGB-D images are clustered into regions and each region is classified between dynamic and static (background) based on the residuals for that region.

**Semantic Information.** Deep neural networks for semantic segmentation can be used to pre-segment image areas that correspond to potentially dynamic objects. While this may certainly help, closed-set vocabularies may be unable to segment all the objects in an image. Additionally, objects being potentially dynamic does not imply that they are actually moving at a particular moment, such as parked cars. For this reason, combining semantic segmentation of potentially dynamic objects with multi-view geometric checks may be convenient [44]. For example, the method

*semantic segmentation*

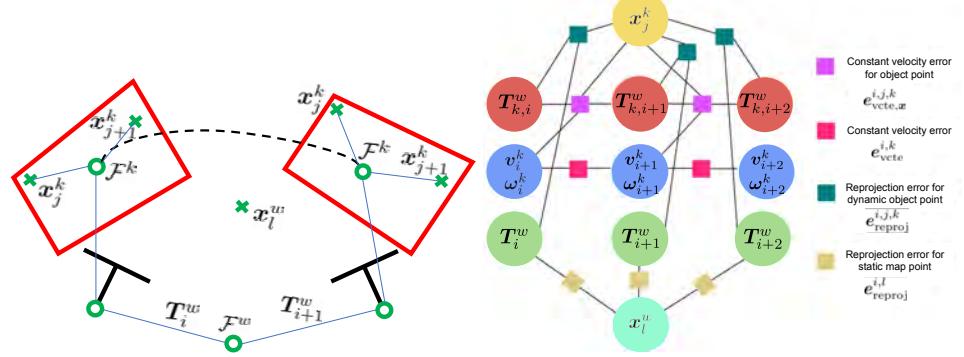


Figure 15.6 Overview of SLAM with dynamic object tracking [76] (©2021 IEEE). Left: rigid body motion assumption. The camera pose  $T_i^w$  is estimated with respect to a static background, while the dynamic object  $k$  is modeled as a rigid body with respect to the world frame  $\mathcal{F}^w$ . Note that all feature points  $\mathbf{x}_j^k$  on each object  $k$  adhere to its rigid body motion  $\Delta\mathbf{T}_{k,i+1}^{k,i}$ . Right: resulting factor graph representation of the problem. The camera poses are connected through odometry factors, while the dynamic object poses are connected through constant velocity factors

MonoRec [1184] advocates the use of a deep network for dense reconstruction from a single moving camera that combines a brightness cost volume —computed from a set of consecutive warped frames— with a mask module that is trained to filter out moving objects. These objects are determined from training data, but also from the color inconsistency across the warped frames, shown in Figure 15.5.

**Domain Knowledge.** Where available, additional prior and domain knowledge can be integrated to disambiguate features of the static scene and dynamic outliers. A common example of this is a no-side-slip constraint for autonomous vehicles. Since we know that a car cannot move sideways, all features that suggest a side-ways motion can be removed as dynamic outliers.

*event cameras*

*radar  
Doppler effect*

**Motion-aware Sensors.** Visual and LiDAR sensors typically provide a capture of the scene at a given time instant, and thus do not provide direct measurements of the motion of dynamic objects. Other sensors, such as event cameras provide a continuous stream of measurements at high temporal resolution. Furthermore, radars and certain LiDARs use the Doppler effect to directly measure the velocity of each observed point. While dynamic SLAM methods using these sensors are actively being investigated, access to such information could be an important cue to identify dynamic objects and their motion. For more details on event cameras and radars, we refer the reader to Chapter 10 and Chapter 9, respectively.

### 15.2.2 Dynamic Object Tracking

Dynamic object removal, as detailed in the previous section, might be convenient if the goal is only to estimate the robot state. However, in many other cases, it may be necessary to track the 3D motion of these dynamic entities, for example in order to navigate without collisions. Note that this problem is highly related or sometimes also referred to as Multi-Object Tracking (MOT) or Multi-Instance Dynamic SLAM (MID). The problem of 2D and 3D object tracking has accrued large interest in the computer vision and robotics communities [1243]. As a result, the body of literature is vast and cannot be exhaustively covered in this chapter. We here thus focus on selected techniques most relevant to SLAM.

The joint problem of simultaneous localization, mapping, and dynamic object tracking can be formulated as follows. The central idea is that each dynamic object in the scene can be modeled as a *rigid moving body*. The factor-graph representation of the static SLAM problem presented in Chapter 7 can therefore be extended to account for individually moving objects, shown in Figure 15.6. As before, most approaches follow a parallel-tracking-and-mapping approach. For each incoming frame, the tracking thread extracts salient features (*e.g.*, ORB [950]). To assign points to individual objects, oftentimes semantic segmentation masks are extracted.

Let  $\mathbf{T}_i^w \in \text{SE}(3)$  be the rigid transformation representing the pose of the camera at time  $t_i$  in the world reference frame  $\mathcal{F}^w$ , and  $\mathbf{x}_l^w \in \mathbb{R}^3$  the coordinates corresponding to rigid map points  $l$ , expressed in the world frame. Each dynamic object  $k$  is represented, at time  $t_i$ , by its six DoF transformation  $\mathbf{T}_{k,i}^w \in \text{SE}(3)$ , which transforms a point from the moving object frame  $\mathcal{F}^k$  at time  $t_i$  to the world frame, and linear and angular velocities  $\mathbf{v}_i^k, \boldsymbol{\omega}_i^k \in \mathbb{R}^3$  at time  $t_i$  in the object frame. Map points  $j$  on dynamic objects  $k$  are represented in the local object frame as  $\mathbf{x}_j^k \in \mathbb{R}^3$ . Note that for rigid-body kinematics the coordinates of points on the object in the local frame  $\mathcal{F}^k$  do not change over time and thus do not depend on  $i$ . This results in the modified reprojection error corresponding to  $\mathbf{x}_j^k$

$$\mathbf{e}_{\text{reproj}}^{i,j,k} = \mathbf{z}_j^i - \pi \left( (\mathbf{T}_i^w)^{-1} \mathbf{T}_{k,i}^w \tilde{\mathbf{x}}_j^k \right), \quad (15.1)$$

where  $\tilde{\mathbf{x}}_j^k \in \mathbb{R}^4$  are the homogeneous coordinates of each object point  $\mathbf{x}_j^k$ ,  $\mathbf{z}_j^i$  is the image point corresponding to  $\mathbf{x}_j^k$ , and  $\pi(\cdot)$  denotes the projection function, here overloaded for points in homogeneous coordinates.

To further constrain the motion of each dynamic object  $k$  through time, a constant velocity model for linear and angular velocity is frequently assumed between consecutive observations of dynamic objects. This can be formalized as the following residual to minimize

$$\mathbf{e}_{\text{vcte}}^{i,k} = \begin{bmatrix} \mathbf{v}_{i+1}^k - \mathbf{v}_i^k \\ \boldsymbol{\omega}_{i+1}^k - \boldsymbol{\omega}_i^k \end{bmatrix}. \quad (15.2)$$

In order to couple object velocities and poses, another error term  $\mathbf{e}_{\text{vcte}, \mathbf{x}}^{i,j,k}$  is added,

*Multi-Object Tracking  
Multi-Instance Dynamic SLAM*

*rigid moving body*

which minimizes the mismatch between the position of each object point  $j$  in  $\mathcal{F}^w$  computed from the motion model and the same position based on the pose estimate

$$\mathbf{e}_{\text{vcte}, \mathbf{x}}^{i,j,k} = h^{-1} \left( \left( \mathbf{T}_{k,i+1}^w - \mathbf{T}_{k,i}^w \Delta \mathbf{T}_{k,i+1}^{k,i} \right) \tilde{\mathbf{x}}_j^k \right), \quad (15.3)$$

where we used  $h^{-1} : (x, y, z, \lambda) \rightarrow (\frac{x}{\lambda}, \frac{y}{\lambda}, \frac{z}{\lambda})$  for the transformation from homogeneous to Euclidean coordinates. The increment in the pose from  $t_i$  to  $t_{i+1}$  is computed from the constant velocity model as

$$\Delta \mathbf{T}_{k,i+1}^{k,i} = \begin{bmatrix} \text{Exp}(\boldsymbol{\omega}_i^k (t_{i+1} - t_i)) & \mathbf{v}_i^k (t_{i+1} - t_i) \\ \mathbf{0} & 1 \end{bmatrix}. \quad (15.4)$$

Figure 15.6 illustrates these modeling assumptions (left) and the resulting factor graph to optimize (right).

This general formulation has found notable success in a number of works. For example, earlier methods such as Multimotion Visual Odometry (MVO) [526] detect tracklets and segment them into clusters such that the rigid-body-motion constraint is satisfied for each cluster. Visual Dynamic Object-aware SLAM (VDO-SLAM) [1267], Dynamic SLAM [448], and DynaSLAM II [76] introduce the factor-graph-based approach and add motion consistency factors between observations. This has been further generalized in AirDOS [894] to capture *articulated objects* consisting of several connected rigid bodies such as human skeletons.

*articulated objects*

*dense reconstruction*

*map representation*

### 15.2.3 Dense Dynamic SLAM

Similar to *static* dense SLAM, discussed in Chapter 5, *dynamic* dense SLAM aims to additionally estimate a dense reconstruction of the scene, however, now including dynamic entities. To achieve this, most approaches separate the process into a *localization* and a *mapping* step.

The localization step aims to track the robot pose with respect to the static background, which can be achieved using any of the sparse dynamic SLAM methods discussed in dynamic object removal (Section 15.2.1) and MOT (Section 15.2.2). Alternatively, also other techniques such as dense tracking applied to the background once dynamic points in the input have been removed can be employed.

In the mapping step, the goal is to build a representation of both the static background and the dynamic entities. An essential design decision is the choice of representation, as this defines which quantities need to be estimated to perform continuous updates of the representation through time. Naturally, reconstructing the background is identical to static dense mapping, however, the same or similar representations can also be used to model dynamic entities. Notice that the representation of the background and dynamic entities do not have to be the same, although in practice this is oftentimes the case for ease of interpretation and processing of the robot map. In this section, we discuss common groups of approaches,

starting from Moving Object Segmentation (MOS) and sensor-level representations in Section 15.2.3.1, to parametric representations if the target object or category is known in Section 15.2.3.2, to reconstructing arbitrary moving objects in Section 15.2.3.3.

#### 15.2.3.1 Moving Object Segmentation

A minimal solution to address dense dynamic SLAM is to detect all dynamic parts, *e.g.*, all points or pixels, in the input data. This problem is often called Moving Object Segmentation (MOS). Similar to dynamic object removal (Section 15.2.1), once these points are detected, they can be ignored to avoid inconsistency and artifacts in the static background reconstruction, or alternatively, be stored as low-level representation of dynamic entities in the scene. To detect dynamic or inconsistent points, similar ideas as before can be applied. However, a central difference is that for dense SLAM methods, also the dynamic point detection needs to be dense, meaning one needs to classify *each* point in the sensor data and not just selected features, as illustrated in Figure 15.7. Therefore, the majority of approaches relies on using geometric consistency and/or semantic information as main motion cues for dense segmentation. In the former, registration residuals can be densely computed during camera-to-model tracking to classify areas of the input image with high residuals as dynamic [981], or one can use the consistency between individual measurements [1251] or measurements fused in the map [972] to identify inconsistent and thus dynamic points. Oftentimes, these detections are complemented with additional clustering steps to deal with noise in the sensing data [981, 1251, 972]. In the latter case, dense semantic segmentation can provide useful information on which parts of the input data reflect classes or instances that are potentially dynamic, such as people, that can be masked out [938]. While semantic segmentation of camera images and LiDAR point clouds has been widely studied and can easily be re-purposed to mask out dynamic classes, classifying dynamic pixels or points directly using deep learning methods on individual or sequences of measurements has become a growing field of research [868, 755, 661]. The learning-based approach has the advantage of being able to incorporate priors about typical moving object shapes, motion patterns, and sensor characteristics, which can improve performance in the target domain, but may not generalize to out-of-domain settings.

*Moving Object Segmentation*

*semantic segmentation*

#### 15.2.3.2 Parametric Models

While discarding dynamic measurements and reconstructing the static background may suffice for certain applications, others, such as manipulation or human-robot-interaction, may require robots to also estimate detailed representations of the dynamic entities in the scene. In these cases, the tracking-by-detection methods of Section 15.2.3.1 have limited expressiveness since they operate on the level of noisy and partial measurements of dynamic regions in each frame.

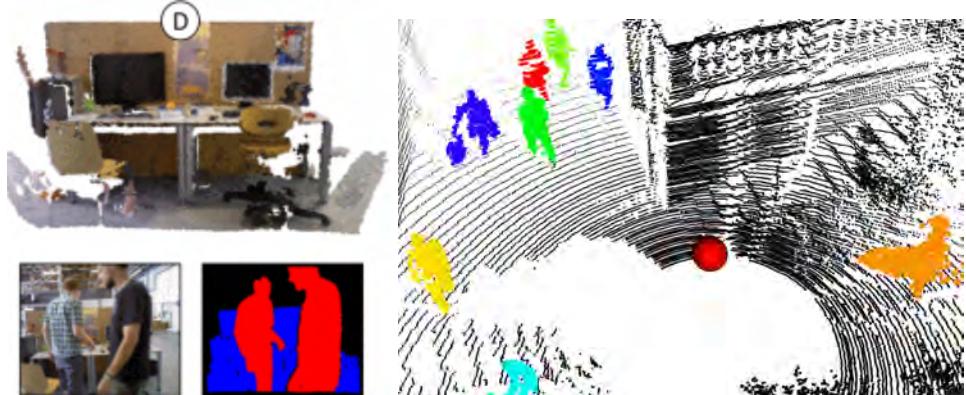


Figure 15.7 Dense moving object segmentation (MOS). (Left) StaticFusion [981] reconstructs the static background (top) from RGB-D images by segmenting each frame (bottom) into dynamic (red) and static (blue) parts based on tracking residuals. (Right) Dynablox [972] densely segments LiDAR scans into dynamic (color) and static (black) points using volumetric map consistency. ([981] ©2018 IEEE, [972] ©2023 IEEE)

*parametric model*

If the type of objects of interest is known, their shape can oftentimes be represented using a *parametric* model. Given this structure, each measurement can be treated as an observation of the true underlying parameters to optimize for them. Similar to MOT (Section 15.2.2), these parameters can be added as variables to a factor graph with model-specific observation factors and estimated jointly with the sensor motion and background. This approach has found particular interest in capturing and tracking human motion, where humans can be represented as skeleton models [894] or dense SMPL [691] meshes [940, 449, 450], among many others.

*rigid scene model*

*direct methods*

#### 15.2.3.3 Simultaneous Tracking and Reconstruction

Alternatively, the area of simultaneous tracking and reconstruction aims to reconstruct arbitrary moving objects. In most cases, one can make the simplifying assumption that each moving object is a *rigid body*. The goal is then to track and reconstruct a single or multiple *rigid moving objects*. The central idea is to create a separate dense representation of each object as well as the background. To achieve this, most approaches follow four main steps [959, 955, 1203, 924]. First, object detection is performed in the input images, typically combining semantic or instance segmentation with geometric refinement and/or geometric motion cues. Second, the camera motion is tracked with respect to the background. Third, each moving object is tracked with respect to the current image. For steps two and three, several techniques are admissible, where typically *direct methods* that minimize the projected depth and photometric residuals of each object are used, since these tracking methods can directly operate on the dense representations of each object

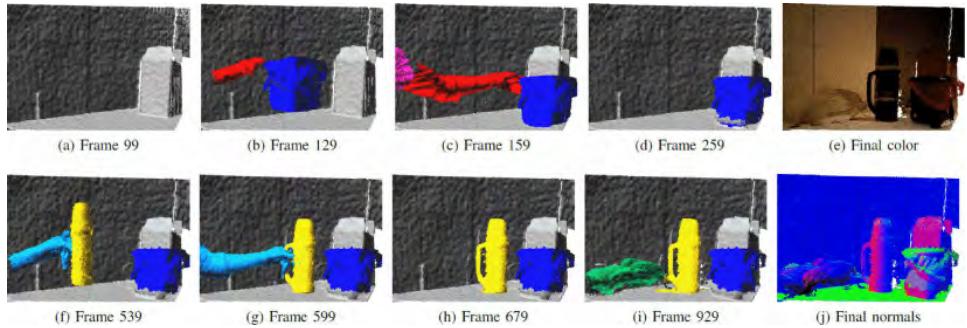


Figure 15.8 Simultaneous tracking and reconstruction with Co-Fusion [959]. Three objects were sequentially placed on a table: First a small bin (blue label), a flask (yellow) and a teddy bear (green). The results show that all objects were successfully segmented, tracked, and modeled. (©2017 IEEE)

or the background. Finally, the measurements of each object and the background are fused into the respective models to incrementally estimate a dense reconstruction of each object. Most prominently, objects are represented as Truncated Signed Distance Function (TSDF) volumes (for more details on TSDF see Section 5.2.2), for their capacity to fuse measurements with sensing noise [959, 955, 1203, 924]. An example of this is shown in Figure 15.8. Alternatively, DirectTracker [386] uses a sparser point-cloud-based reconstruction to jointly estimate the object grouping and respective rigid body motions for each object.

The simultaneous tracking and reconstruction approach has the large advantage of being more general as no prior object models are needed. However, the many steps required can be computationally expensive, especially for large scenes and in the presence of many dynamic objects. A further limitation is the fact that errors in object tracking can lead to misaligned measurements being fused into the object representations, which can degrade the reconstruction and further inhibit accurate tracking in the future, where, for example, keypoint-based tracking methods such as BundleTrack [1176] can be more robust.

The assumption of objects being rigid can further be relaxed to articulated objects by estimating transforms for each rigid part connected to a complete object [974], or to general deformable objects. Examples of the latter include DynamicFusion [799] and KillingFusion[1013], which will be covered in more detail in Section 15.4. An example of a learning-based approach is AnyCam [1186] shown in Figure 15.9, which makes use of a transformer-based neural network in order to jointly estimate the camera motion and a 4D reconstruction of a dynamic scene with non-rigidly moving objects from a casual monocular video.

Finally, the additional reconstructed objects could also be used to improve the tracking of the robot’s ego-motion. However, this only appears to improve the robot

*Truncated Signed Distance Function*

*articulated objects*

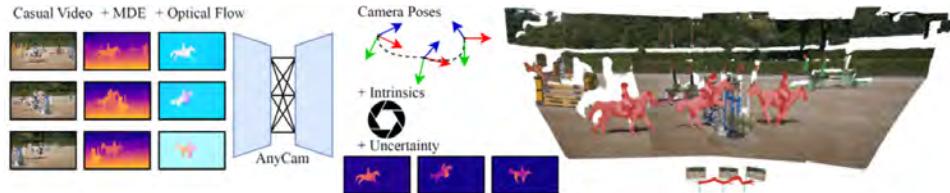


Figure 15.9 AnyCam [1186] is a transformer-based neural network that jointly estimates the camera motion and a 4D non-rigid reconstruction from a casual input video. (©2025 IEEE)

state estimates in corner cases, where tracking against the background is insufficient, for example when large parts of the robot’s view are occluded ( $>70\%$ ) and good motion models for the moving objects are available [688].

### 15.3 Long-term Dynamic and Lifelong SLAM

The previous section addressed considerations to handle *short-term* dynamic observations. In this section, we will address the challenges and implications of *long-term* dynamics on SLAM.

*long-term dynamics*

*data association*

*perceptual aliasing*

In general, there are several notable challenges for long-term dynamic SLAM. First, in contrast to short-term dynamics, the observed changes’ patterns are more abrupt, and can in principle grow *arbitrarily large* (especially the longer a robot does not observe a space the more likely it is to be ever more different from when the robot last observed it). This makes *data association* significantly more difficult. For example, whereas short-term motion is, by definition, small between observations and data association can be addressed through tracking, data association across long-term dynamics may require solving a potentially global re-identification and matching problem. Furthermore *perceptual aliasing* is exacerbated, as, for example, a unique landmark (or several if they are feature points on a single object) can be observed in several positions because it has moved.

Second, because the magnitude of changes is larger, many effects —that can potentially be neglected in the short-term case— become significant in the long-term case. This includes geometric changes, from motion (such as objects being added, moved, or removed from the scene [971]) to structural changes (for example, consider a construction site that will look fundamentally different at each visit [1054]), as well as appearance changes from illumination (such as from different weather, time of day, or seasons [1166]) to changes in texture or reflectivity of surfaces.

Finally, because long-term dynamic observations typically coincide with longer elapsed times and farther robot motion, large changes in viewpoint as well as potential sensor degradation or change between observations can occur.

To address these challenges, we will first briefly consider implications for state

estimation in Section 15.3.1, followed by offline approaches for persistent scene representation in Section 15.3.2. Extensions thereof for online SLAM and unified short and long-term dynamic SLAM will be covered in Section 15.3.3. Finally, we will introduce temporal scene understanding models in Section 15.3.4, which aim to go beyond capturing spatial variations to reconstructing temporal patterns.

### 15.3.1 Lifelong SLAM for Localization

A central capacity of lifelong SLAM systems is the ability to accurately localize the robot in spite of the challenges pointed out above. This problem is conceptually very similar to the *place recognition* or *loop closure detection* problem, which addresses the related problem of identifying whether two views are of the same place in spite of changes. In this section, we expand these ideas for continuous long-term localization. While a range of approaches exist, we will summarize only a few of the most relevant ones.

#### 15.3.1.1 Parallel and Summary Maps

A first group of approaches to this problem, similar to the case of dynamic object removal in Section 15.2.1, leverages the fact that the robot pose can often be estimated from a subset of inlier points that are matched. However, instead of removing points as outliers, the idea here is to keep adding additional feature points to the map if their appearance has changed to the extent that they can no longer be matched to their previous observations. Typically, individual visits to a scene are referred to as *experiences* or *sessions*, where the changes within each experience are small enough to enable tracking of the robot pose [219, 849]. If the current experience can be localized with respect to previous ones, this implies that the map is expressive enough to localize the robot, and no further data needs to be added. If localization is weak or fails, the new data is added to the map. This has the advantage that new points are only added when needed and the complexity of the map thus reflects the variation of appearance in the scene [219]. The localization performance with respect to a single reference frame can further be improved by adding cross-experience constraints. For example, experience  $\mathcal{E}_A$  may not be able to sufficiently localize w.r.t.  $\mathcal{E}_C$ , but if both can localize to an intermediate experience  $\mathcal{E}_B$ , the constraints  $\mathcal{E}_C \rightarrow \mathcal{E}_B \rightarrow \mathcal{E}_A$  can refine the map coherence and localization accuracy [296, 777, 849]. To address the problems of many experiences leading to a possibly large map and potentially misleading features (such as from moved objects) remaining in the map, maps are sometimes optimized and *summarized* offline, running complete bundle adjustment (BA) between all experiences and retaining only the features most likely to aid future localization [777, 296].

### 15.3.1.2 Appearance-Invariant Representations

Instead of mapping features for each appearance, recent work focuses on feature detection, description, and matching algorithms that allow a single feature to be matched across varying appearances. This has primarily been made possible through advances in deep learning, that allow (i) training such methods on large datasets covering a range of different conditions and (ii) taking much more context into account (*e.g.*, up to entire images compared to pixel neighborhoods for classical methods) [968, 35, 267].

A special kind of appearance-invariant feature to highlight is semantic information. For example, if one can identify that an object is a chair in different visual settings, the fact that it is a chair is not going to change over time. Therefore, the presence of specific objects and their configuration can be used for localization across large visual changes, such as aerial and ground viewpoints [369].

Once such appearance invariant features and descriptors are extracted, they can be integrated into the SLAM pipeline as landmarks to facilitate map optimization and long-term localization. While deep learning-based approaches have shown strong performance, they typically require a GPU which may not be available on all robots and may be computationally more expensive than classical features. Finally, while this can address some challenges related to appearance, outlier matches such as objects that have moved or changed still need to be identified and rejected.

### 15.3.1.3 Memory, Scaling, and Marginalization

A fundamental problem for robots that operate over long times, and potentially indefinitely, is that information keeps being collected by the robot and added to the map, eventually resulting in computer memory and processing time limitations.

**Forgetting.** One strategy to partially mitigate this is to incorporate mechanisms for *forgetting*. Typically, this is implemented with a weight or probability of persistence of map points that diminishes over time, such that points that have not been observed for a long time have a lower probability of corrupting current localization as outliers and eventually are pruned from the map [933, 262]. Alternatively, *map summarization* [296, 777] can be run, oftentimes as an offline process, to retain only the features most likely to support future localization. One advantage of this approach is that it provides an ordering of importance such that only the top-N points can be kept subject to a memory budget constraint. This allows a robot to keep as much information around as it can fit in its memory and only start forgetting afterwards.

**Memory Management.** Instead of forgetting, another option is to dynamically save and load map features in and out of the memory of the robot. An example of this is shown in RTAB-Map [614], where nodes can be moved from short-term to working to long-term memory and back, ensuring that the number of nodes currently being processed is small enough for real-time computation. This prevents

the need to completely remove nodes, but incurs additional computational cost for memory management and may eventually run out of capacity.

**Marginalization.** In addition to the number of points in the map, pose-graph-based SLAM methods optimize over the history of robot poses, which usually grows linearly with elapsed time. To avoid infinite growth in the number of nodes and factors, the process of *marginalization* aims to remove old nodes *without* losing information. The central idea is to replace a set of nodes (or other information, more generally) with a single node whose properties reflects or summarizes the information previously stored in the replaced nodes. In probabilistic models, this is the *marginal distribution* over the removed nodes (see *e.g.*, [255], Sec. 5.3 for more details). This allows effective compression of the graph and reducing the size of the problem without discarding nodes completely. Nonetheless, in practice, some information is lost when marginalizing nodes. Most importantly, the topology of the marginalized nodes can no longer be changed. For example, if there is uncertainty whether certain factors, such as observations or more importantly loop closure detections, are inliers or noise, marginalizing them will freeze their current assignment into a single node which cannot be undone, even if future measurements arise that would contradict or disambiguate this assignment. In summary, marginalizing previous measurements provides an effective strategy to reduce the size of the problem while retaining information, but reduces the flexibility of the optimization by “baking in” errors that cannot be easily corrected afterwards. Thus, when to marginalize is an important open question.

### 15.3.2 Map Cleaning and Change Detection

Even with eventual forgetting of points or objects, detection-based methods such as the ones introduced in Section 15.3.1 tend to accumulate outdated measurements (for example of objects that have since moved). Specifically, when a detector fires it is evidence that an object or feature is present and can be added to the map. However, the inverse is not necessarily true. This manifests in the *absence of evidence* vs. *evidence of absence* problem. For example, consider a robot that visits a room and observes a chair in a corner (positive detection). If the robot later revisits the room and does not observe the chair, this might be the result of the robot not looking into the same corner and the chair might still be there (absence of evidence), or of the robot looking into the corner and establishing that the chair has disappeared (evidence of absence, or *negative* detection). In addition, since detectors are imperfect, it is also possible that the robot looked into the corner but did not recognize the chair, *e.g.*, due to illumination changes, in which case the conclusion that the chair is now absent would also be incorrect.

The establishment of such negative observations, or changes, is the goal of *change detection*. Historically, this has mostly been addressed in an offline or post-processing setting, since the goal is oftentimes to retain a static map after data collection by a

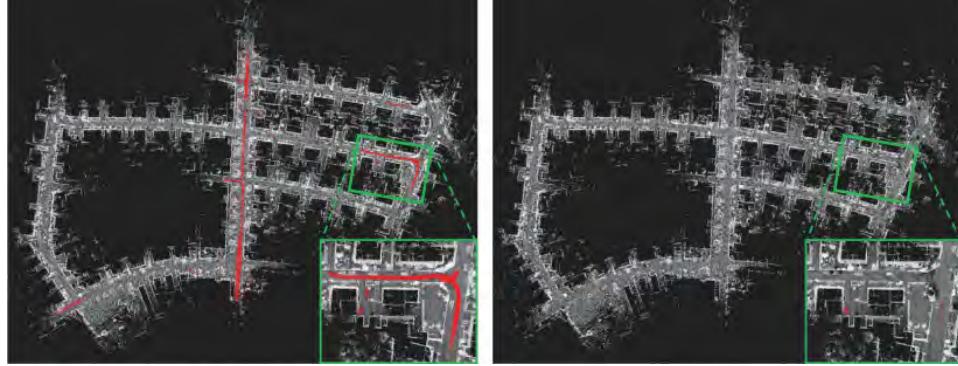


Figure 15.10 Example of *map cleaning* by ERASOR [660]. Path artifacts from measurements on dynamic objects (red) are largely detected and removed from the static map. (©2021 IEEE)

robot. We will further detail this setting in this section and discuss recent extensions to online SLAM in Section 15.3.3.

#### 15.3.2.1 Map Cleaning

The goal of map cleaning is the removal of dynamic and spurious measurements from a map in order to only retain the high-quality, static, and persistent features of a scene. The most common use case is to filter out spurious points and observations of short-term dynamic objects from a mapping session [563, 660], but the same techniques can also be applied to long-term and multi-session mapping [879].

Conceptually, the underlying detection mechanisms and principles are very similar to *dense dynamic point detection* discussed in Section 15.2.3.1. However, for offline processing, additional resources are available. In a first step, globally consistent robot poses are estimated through robust BA, where dynamic and spurious points are ignored as outliers [563, 660]. Once the sensor poses are fixed, all measurements are revisited to ensure consistency and remove spurious data. Similar to Section 15.2.3.1, prominent methods include fusing all measurements in a *volumetric map* of the scene (typically through ray-casting into an occupancy or TSDF voxel grid), where the consistency of measurements can be checked in each map cell [970]. While this considers all data, it can also lead to large memory consumption and possibly long processing times. To avoid this, *visibility-based* methods, instead of a globally fused map, select a subset of nearby measurements to compute the consistency of points across them, oftentimes directly on the sensing data [563, 660, 879].

*voxels*

A notable advantage of offline processing is that all measurements are available to use in map cleaning, and that a variety of additional steps, such as ground plane segmentation [660] or iterative algorithms to incrementally add high-confidence detection [563], can be performed. An example of this is shown in Figure 15.10.

### 15.3.2.2 Change Detection

Historically, *change detection* is a term that has been used in a variety of communities. Most prominently, *2D* or *image-based* change detection has been widely studied in computer vision. In this setting, the goal is to identify differences between two images of the same scene or even from the same view-point, with applications in background subtraction, surveillance, medical imaging, and many others [897]. Additionally, image-based change detection from satellite data is an active area of research in remote sensing [39].

*change detection*

In robotics, the ability to detect changes between images is also highly useful. However, in the context of mapping, where the goal is to estimate the underlying *3D structure* of the scene from all observations, change detection can directly be performed in 3D at the level of submaps or sessions. This is especially useful if measurements are eventually marginalized or fused in the map and may no longer be available. Although interest in image-based change detection for robotics has recently re-surfaced with advances in high-quality novel view synthesis [703], we will primarily discuss 3D and mapping-based change detection approaches in this section.

It is important to note that the distinction between map cleaning and change detection can be fuzzy, since they address related problems. In the most typical case, change detection is posed in a *multi-session* mapping scenario (for example, consider a robot that scans a room in the morning and again in the evening), where each session allows the reconstruction of an internally consistent (possibly cleaned) map. The goal is then to identify and capture changes between the two visits on the scene level. To achieve this, oftentimes detailed scene representations are required and *dense* reconstruction methods are used. In addition, meaningful change representations frequently require object-level reasoning, where an object can be considered a *unit of coherent change*. Most prominently, this includes objects (such as mugs or chairs) being added to, removed from, or moved around the scene, which will be the focus of this section. However, this could in principle also include many different and non-geometric changes, such a cushion being deformed, a wall being painted differently, or button being switched on or off.

*Multi-session Mapping*

*dense reconstruction*

**Geometric Change Detection.** As in earlier sections in this chapter, a number of cues can be used to contrast measurements, which can also be applied for change detection [564]. To contrast maps directly, however, *volumetric* representations such as occupancy or TSDF maps have proven most useful for change detection [321, 948]. This is primarily due to the fact that, in contrast to solely *dense surface* representations, they explicitly differentiate between *unobserved* and observed to be *free* space, which is essential to disambiguate the *absence of evidence* problem. During change detection, the two volumetric maps can be compared against each other in a process called *volumetric*, *map*, or *scene differencing*, where surfaces previously observed to be free must have newly appeared, and old surfaces now

*Truncated Signed Distance Function*

*scene differencing*

observed to be free must have disappeared. Finally, areas only observed in one of the sessions can be included to complete the map, and surfaces observed twice can be fused to increase the accuracy of the reconstruction.

To create consistent maps and avoid false positive detections, one first needs to accurately register all submaps or sessions into a common reference frame. Since there may be odometry drift or inaccurate state estimates within each session, it is often not sufficient to just rigidly align the individual sessions, but best results are achieved by jointly optimizing for alignment between sessions, typically through inter-session constraints and BA [321, 948, 564].

#### *semantic segmentation*

**Semantic Information.** In addition to geometric checks, semantic information is frequently incorporated for change detection for a number of reasons. First, since the semantic label of an object is appearance invariant, it can provide a powerful abstraction to compare submaps created under different conditions. Note that the use of semantics does not eliminate the problem of appearance changes, but simply offloads them to a detection network which can be trained on large datasets, or specialized detectors for different sensors and conditions can be used. Second, semantics can provide valuable non-geometric information, for example a sheet of paper being removed from a table will not change the geometric surface, but may be easy to detect semantically. Finally, the goal of change detection is often to identify objects as a *unit of coherent change*. Since coherent motion is also an essential aspect of human scene understanding, there is large overlap between this definition of an object and most closed-set semantic labels (for example mugs, chairs, and similar objects often tend to move as a unit). To achieve best performance, many systems combine geometric and semantic information [623, 971].

#### *partial observations*

**Semantic Consistency.** The problem of *semantic consistency* arises from the combination of the facts that scenes typically change in *coherent units*, but robots can generally only obtain *partial measurements* of the scene. For example, consider the case where a robot observes a table. Later that day, it returns and sees that the table has disappeared, but half of the view is occluded. If change detection is carried out on the observations or maps of that scenario, only the observed half of the table will be removed leaving half a table floating around, whereas for a human this outcome would seem rather implausible (*i.e.*, we have a strong prior that tables tend to move as a unit). This problem is illustrated in Figure 15.11. To avoid such artifacts, one can forget previous maps and map every time from scratch, which will ensure semantic consistency but also prevent the robot from ever building more complete models of the scene. More commonly, change detection can be performed at the level of objects or semantic classes to try to prevent such artifacts from forming. Finally, future observations may also remove artifacts as changes once they are observed, but the presence of artifacts in the map may reduce the overall map fidelity and hinder its use for planning and autonomy.



Figure 15.11 Illustration of the *semantic consistency* problem. In this scene, the sofa and lamp were switched and objects on the table were changed between two robot observations. The left image shows the reconstruction when all measurements are fused, whereas on the right semantic-aware change detection is performed [971]. Since the sofa was only partially observed on the second visit, we see parts of it remain on the left whereas the entire sofa is removed on the right (shaded red). Furthermore, conflicting measurements of free space and the sofa are fused on the left, leading to only partial reconstruction of the sofa in the new place. Finally, objects with small geometric changes such as the journal on the table are not recognized and merged with other observations. (©2022 IEEE)

#### 15.3.2.3 Object Understanding through Changes

**Object Detection from Changes.** The inherent relation between changes and semantic objects can be used to make deductions in both directions. While we have seen above how semantic information can be used to improve change detection, the fact that something has changed can also be used as a cue for object detection. To this end, typically, changes in a dense reconstruction are detected to identify all moving surfaces. These can then be spatially clustered to identify likely objects [321, 28, 327]. This complementarity between geometry and semantics is particularly useful if no semantic object detector is present, or to complement semantic segmentation by detecting objects that were not correctly segmented. However, as before, geometric object segmentation alone is susceptible to partial observations. For example, if two touching objects were both removed they will likely be under-segmented into a single object. Similarly, an object partially observed through occlusions will likely be over-segmented into several parts.

**Object Instance Re-localization.** In order to build an object-centric understanding of the scene through time and not only an estimate of the current or static state, the goal of object instance re-localization is to track objects through long-term dynamic observations. Conceptually, this problem and most solution approaches are again similar to the ones for *place recognition* and *loop closure detection*. However, there are a number of additional considerations that make this a challenging problem.

First, by the abrupt nature of long-term dynamics, objects can, in principle,

*object re-localization*

*place recognition*

change or move arbitrarily far between observations. Of course, based on the time elapsed and the motion model assumed for an object, the last observed state of an object may be an informative cue to associate it to current observations [948, 96], but this may in general not always be the case.

Second, since an individual object is much smaller than a scene, there is often less variety in geometry and texture, and thus many fewer features that uniquely describe it. This challenge is exacerbated for small objects by the fact that if they are not observed up closely, fewer and lower resolution measurements are available to extract features from.

#### *perceptual aliasing*

Third, since there may be several instances of identical or similar object types in a given environment, *perceptual aliasing* is a major challenge. This further raises a more philosophical question of what it means for an object to be a unique instance and whether they should be individually tracked. For example, consider a meeting or seminar room with tens or hundreds of identical chairs. In this case, it may be hard to tell which chair is which, and how each of them was moved over time. However, this might also not be relevant information for most tasks, and it might be preferable to fuse all observations of any chair into a single representative model that is replicated many times. On the other hand, if you consider the case of a coffee mug, there may be numerous identical mugs in an office or building, but people might care a great deal to keep track of their own mug. One option to address the challenge of perceptual aliasing is to take the uniqueness of solutions into account, *i.e.*, unique matches of a highly distinct object can directly be accepted, whereas for objects with several similar match candidates, a *multi-hypothesis* approach or adequate *uncertainty measure* can be employed.

Finally, one can consider the object instance re-localization problem in a *closed set* or *open set* setting. In the former, one assumes that the set of objects in a space is fixed, and re-localization essentially becomes an optimal matching problem. In the latter and more general case, objects can also be added or removed from the scene, which raises the additional question of “when are object observations similar enough to be the same,” reflecting the fact that there is always the alternative explanation of a new but similar object having appeared in the scene.

To work towards these challenges, prevalent solution approaches primarily focus on descriptor extraction and matching. One family of methods focuses on extracting a set of keypoints on each object and descriptors for each object, which can be directly used to identify correspondences and solve for 6DoF transforms. As a natural first step, such keypoints could directly be visual features from the SLAM frontend [745]. Alternatively, 3D keypoints and descriptors such as Fast Point Feature Histograms (FPFH) [957] and Signature of Histograms of Orientations (SHOT) [962] have found significant use for changed object re-localization. These have the advantage that they can be extracted from the dense 3D reconstruction of a session and are insensitive to variations in appearance. However, since they only represent a local patch of geometry, they tend to be less descriptive

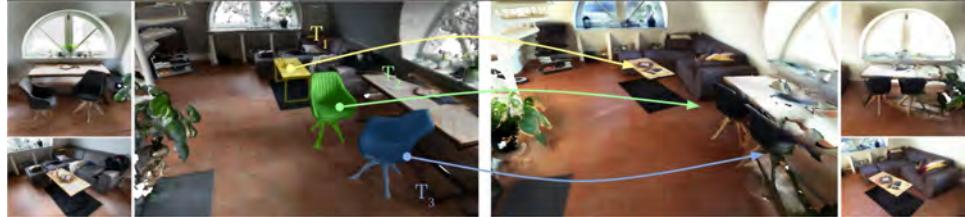


Figure 15.12 Object instance re-localization task in the RIO dataset [1137]. The goal is to identify which object instances correspond between an initial scan (left) and later re-scan (right), and estimate the 6-DoF poses  $T_1$ ,  $T_2$ , and  $T_3$  between each object and its new location. (©2019 IEEE)

and often require further geometric verification [351]. Recently, deep-learning-based local shape (and sometimes appearance) descriptors achieve the best performance since they can be trained to emphasize the most informative aspects of a shape or object [1137]. An example of this is shown in Figure 15.12.

This has also given rise to a second family of approaches that compute a *single descriptor per object* instead of several keypoints [948, 1304]. This method is advantageous as keeping track of a large database of keypoints and exhaustively matching them can be computationally expensive. An important aspect of these methods is to construct or train the networks such that they are SE(3)-*invariant* or SE(3)-*equivariant* to allow matching across different viewpoints and registration of the matched objects, respectively.

Finally, once the many partial observations of changed objects can be re-localized and registered, this not only provides valuable insights about the history of each object, but also allows the reconstruction of ever more complete object models and richer scene descriptions over time [351, 1304].

### 15.3.3 Change-aware SLAM

While multi-session mapping provides an exemplary case of long-term dynamic observations, it is by far not the only setting where they can occur. In practice, such changes may occur over much shorter time scales. For example, consider a robot that moves from a living room to the kitchen to fetch something. If the living room changes in between, it does not matter whether the robot returns a week or only few minutes later. At its limit, this can be as fast as a robot or sensor just looking away and finding the scene changed when looking back. The goal of change-aware SLAM is to be able to detect and represent such long-term changes during *online operation* of the robot or system.

**Challenges.** Compared to *multi-session* mapping, where all changes are assumed to occur between sessions, changes can now occur at any point between observations.

As a consequence, assumptions about stationarity can only be made within a short window, and less data is available to fuse and disambiguate noisy observations. Furthermore, the perceptual similarity between inaccurate relative localization and changes in the observed scene can make it difficult to disentangle the two problems. As an example, if the robot’s pose estimate is offset by a meter, the surfaces of previously mapped objects may appear to be absent in the current, offset view, even though that is not the case. This interplay between change detection and localization, *i.e.*, the classification into inliers and outliers for each localization constraint, is further exacerbated through the influence of one’s solution on the other. This potential problem is well illustrated if we consider the two extremal solutions: if all measurements are considered unchanged inliers in spite of possibly large changes in the scene, the resulting registration solution may be inaccurate and represent an “average” of true and false measurements. On the other hand, if the entire previous scene is considered changed and now absent, the current open-loop estimate is trivially optimal and the resulting pose estimates and scene reconstruction may not be complete and consistent. Finally, change detection has to operate continuously and is based on the partial data collected by the robot so far. This requires that change detection can run at higher rates in order for the robot to have an up-to-date understanding of the environment, and ideally that false inlier/outlier change decisions can be corrected as more data is collected.

#### *change detection*

**Object-level Change-aware SLAM.** Since global reasoning about changing scenes on low-level sensor data quickly becomes intractable, the vast majority of change-aware SLAM methods first locally summarize measurements. Due to the nature of most considered changes happening at the level of rigid semantic objects, object-level representations are an intuitive choice [971, 888, 345, 889, 973]. In this setup, an object-level scene representation is extracted, where localization and reasoning about changes happens also on the level of each object, presenting a much more manageable problem than reasoning about individual sensor measurements, and, by construction, enforcing semantic consistency of the scene representation.

**Local Consistency.** An important consideration in change-aware SLAM is how to disambiguate noisy measurements from actual changes in the scene, and consequently, how these measurements are integrated into the map. For example, if change detection is run on the raw data of every frame, some noisy points may penetrate into existing objects and falsely mark them as changed. On the other hand, if more conservative change detection rules are employed, it is easy to get false negatives and fuse together data that are not observations of the same object, leading to corrupted and inconsistent reconstruction (see Figure 15.11). To overcome this, the central idea is to use *local consistency* to establish sequences of measurements that can be guaranteed to be change-free. At a minimum, this is the case when measurements are tracked frame-to-frame [971, 1127]. Intuitively, this corresponds to the fact the scene cannot change unobservedly while the robot is observing it. In practice, through adequate assumptions on the rate of change in

the scene and the odometry error being low over short times, this can be extended to a sliding [345] or active [973] window.

**Change-aware Mapping.** Combining these ideas, a first real-time change-aware mapping system is presented in Panoptic Multi-TSDFs [971]. Assuming globally consistent poses are given, the Panoptic Multi-TSDF frontend estimates a set of semantically and locally consistent submaps by only updating submaps that can be tracked frame to frame, called the *active* submaps, and for which the semantics of the measurements matches the target submap. By differencing the active to overlapping inactive submaps, changes can be efficiently detected on this higher-level abstraction of submaps, and previous submaps can be discarded or merged if they disagree or agree with the current submaps, respectively. This allows retaining information from previous submaps where they match, and by removing entire submaps enforces semantic consistency by construction. An alternative approach is presented only shortly after in POCD [888], where incoming frames are also segmented into objects, but then are tracked and registered against the current map. This has the advantage that some level of odometry noise can be tolerated and corrected. To handle noisy change observations, a probabilistic persistence model for each object is created, and is updated when an object is observed, or an object that should have been observed is not registered, respectively.

**Globally Consistent Change-aware SLAM.** In large-scale scenes and when no global poses are available, these have to be jointly estimated with the scene changes. For object-level representations, this can directly be done using a factor-graph-based approach with the partial, locally consistent object observations acting as landmarks. To deal with the problems of data association in change detection, NeuSE [345] presents an approach to learn highly descriptive and SE(3)-equivariant neural object representations. These latent descriptors capture the complete shape from partial observations and allow for a direct association and registration of object instances. As a result, such associations can be used as loop closure constraints and spatially-inconsistent or missing associations used as a change signal. Alternatively, POV-SLAM [889] presents an approach using the expectation minimization (EM) algorithm to iterate between optimizing the poses of the robot and objects given the associations between object observations, and optimizing the object observations and persistence probabilities given the spatial layout of the scene. This allows it to continuously refine both localization and change detection, especially when new observations are made in the future. Instead, Khronos [973] creates candidate association factors between nearby objects and uses robust factor graph optimization via graduated non-convexity [1219] to determine the inlier associations. Furthermore, the (assumed static) background deformation is incorporated as extra factors to aid spatial optimization, and a deformable geometric change detection step is performed after loop closure to resolve the evidence of absence vs. absence of evidence problem.

*submaps*

*scene differencing*

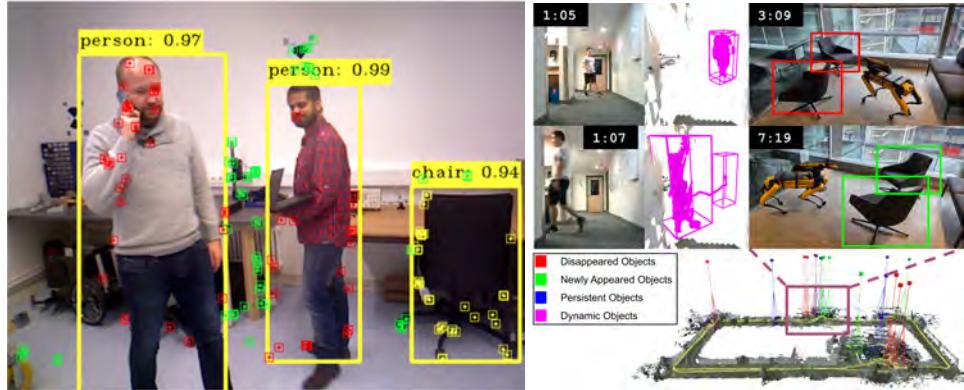


Figure 15.13 Unifying short and long-term dynamic SLAM. Changing-SLAM [1127] (left, ©2023 Springer) tracks keypoints for short-term dynamic objects (red), the background (green), and associates potential long-term dynamic objects (yellow). Alternatively, Khronos [973] (right) densely detects and reconstructs short-term motion (purple) and long-term changes (removal and addition shown in red/green) during online operation.

**Unified Short and Long-term Dynamic SLAM.** Finally, the notion of local consistency can be used to combine short-term dynamic object tracking with change-aware methods in order to capture various dynamics patterns at the same time, shown in Figure 15.13. A first formulation of this is presented in Changing-SLAM [1127]. Changing-SLAM utilizes a sparse SLAM formulation building on top of ORB-SLAM [784], where in each frame feature points are grouped into objects by semantic masks. These points are then tracked frame-to-frame using a Kalman filter to capture short-term motion. Object points that are not tracked (long-term changes) are associated globally to nearby objects of the same semantic class. Finally, a persistence score is estimated for each object such that objects that have not been detected are forgotten and removed from the map.

A first method for dense spatio-temporal metric-semantic SLAM is presented in Khronos [973]. Following similar ideas, Khronos uses a volumetric local map to extract reconstructions of the background and tracked static and moving objects. After optimizing for globally consistent poses after loop closure, an additional change detection step estimates when each object change occurred to estimate the history and evolution of the state of the scene.

#### 15.3.4 Temporal Scene Understanding

When mapping an environment, the goal is to build a digital representation and understanding of the scene. As discussed before, when mapping dynamic environments, the desired level of understanding can vary from building a model of the

*persistent* parts of the scene —excluding motion and changes— to building a detailed *4D* reconstruction and understanding of what moved and changed, when, and how. However, such an understanding can go beyond capturing the individual moving and changing entities to estimating the underlying *temporal variation patterns* and predict the future evolution of the scene. In this section, we briefly overview three key families of temporal models.

**Maps of Dynamics.** While *dynamic maps* focus on detecting and tracking moving objects, Maps of Dynamics (MoD) take these trajectories or other motion information as input to estimate the *typical motion patterns* in the scene [608]. Such patterns can be constant (such as the dominant motion direction in a one-way road), or also time dependent (for example, people tend to move into the office in the morning and move out again in the evening). Understanding these patterns can provide valuable information for navigating dynamic scenes or human motion prediction.

*Maps of Dynamics*

**Periodic Events.** A special but commonly found kind of dynamics pattern are periodic or cyclical events, such as for example daily or yearly variations of a scene. If one assumes that the state of a scene is governed by an underlying hidden process that is periodic, observations of the scene can be treated as observations of the periodic process and used to estimate its properties. This idea has been pioneered by Krajnik *et al.* [602], who propose *frequency maps* to analyze the different periodic processes of observations in the time domain through spectral analysis in the frequency domain by means of the Fourier transform. By only storing the dominant modes of the frequency spectrum, a compact map representation is achieved that can capture periodic events of different (and arbitrarily long) frequencies. Since the complete *4D* model of the scene is defined by these modes, once they are estimated from past observations, all future states of the scene can be predicted, which can improve future localization and path planning [601].

*frequency map*

**Learning-based Methods.** Beyond structured or manually designed statistical models of dynamics, learning-based methods can also be used to predict future variations in the scene. Typically, object-level changes or motion are predicted using scene graphs as compact representations of objects and their relations to their surroundings [848, 690, 393]. The learning-based approach has several advantages, including higher flexibility to represent complex dynamics patterns and the ability to combine semantic priors (*e.g.*, that dining tables tend to change around noon due to lunch) with observations from a given environment. This can lead to better prediction performance compared to more static co-occurrence priors or pure frequency-based models [848, 393], and improve active and pro-active robot behavior in the future [848, 690]. As a disadvantage, more data may be necessary to train more complex models, where change data may be scarce and imbalanced with respect to static observations of the scene.

*scene prediction*

## 15.4 Deformable SLAM

Beyond rigidly moving and changing objects, robots frequently operate in environments that are not only dynamic, but also contain *deformable* entities. If only parts of the scene are deforming and should be modeled, the robot motion is typically estimated with respect to the static background as introduced in Section 15.2.3, simplifying the problem to *deformable reconstruction* with known sensor poses. This problem can be addressed as Non-Rigid Structure from Motion (NRSfM) and will be covered in Section 15.4.1.

In the more general case, *Deformable SLAM*, or SLAM in deformable environments, refers to SLAM in environments *without a rigid background*, i.e., in which there is not any visible rigid or static part. A particularly relevant case is in minimally invasive robotic surgery scenarios, where the only available sensor is a monocular camera that can only observe the deformable organs inside the human body [890]. This setting requires the estimation of the ego-motion of the camera with respect to the constantly deforming scene. Since the lack of any static parts makes the deformable SLAM problem severely under-constrained, it is a more challenging problem than static and dynamic SLAM.

In the following, we start by discussing the differences between deformable SLAM and NRSfM in Section 15.4.1. We then first present the simpler case where depth information, for example from an RGB-D camera or stereo cameras, is available for deformable SLAM in Section 15.4.2. Second, we introduce the pipeline for deformable SLAM for the challenging case of monocular cameras in Section 15.4.3, followed by details about map initialization and map extension in Section 15.4.4.

### 15.4.1 NRSfM vs. Deformable SLAM

Similarly as rigid visual SLAM is connected with SFM, non-rigid SLAM has a strong connection with its non-rigid counterpart NRSfM.

#### 15.4.1.1 Non-Rigid Structure from Motion (NRSfM)

*Non-Rigid Structure from Motion (NRSfM)* NRSfM is a classical topic in computer vision which addresses the problem of modeling deforming 3D environments using monocular vision, and provides the theoretical foundations for understanding the 3D modeling of a deforming scene from a video sequence. Each 3D point in the scene has a distinct 3D position in every frame, therefore, the map does not represent a single position but rather a 3D trajectory for each map point. As a consequence, the monocular problem is severely under-constrained and additional priors are required to constrain the solution. Popular options are *temporal smoothing* priors, which assume that the trajectory of a 3D point is smooth over time, and *spatial* priors that assume that the deformation of a 3D point is similar to that of its neighbors.

The classical formulation of NRSfM assumes a static monocular camera observing

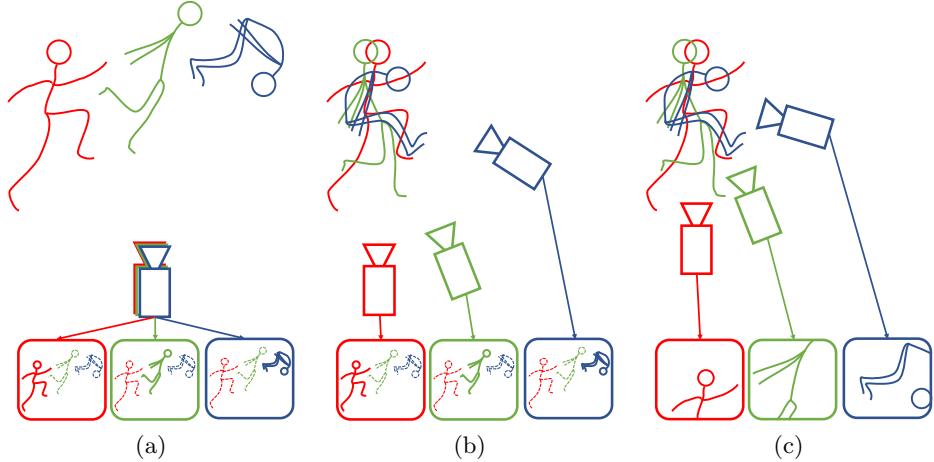


Figure 15.14 Sketch of NRSfM vs. Deformable SLAM in 3 frames. The scene *does not include any rigid background*. (a) The classical NRSfM assumes a fixed monocular camera while the 3D object undergoes a non-rigid motion which includes a rigid transformation and a local deformation. (b) Deformable SLAM assumes that the scene object mainly undergoes a non-rigid motion while the moving camera focuses on the rigid motion despite the absence of a static background. (c) Deformable SLAM also often assumes that the camera is in a close-up, where the deforming scene is only partially observed, *i.e.*, the camera undergoes an *exploratory trajectory*.

a moving object without any static background. Most of the video frames observe the full object extent, *i.e.*, most of the images overlap while observing the same deforming object. Typical cases are a moving hand, a waving flag, or a jumping person, all in front of a camera. It is important to note that despite the camera being fixed, the motion of each point often includes a significant rigid transformation of the object combined with a local deformation thereof. The rigid body motion is never explicitly computed, but included in the non-rigid scene motion of each point with respect to the fixed camera (see Figure 15.14(a)). Although the rigid motion is not computed, it implies a significant parallax that makes the estimation problem well conditioned.

**Shape from Template.** A first family of NRSfM are Shape from Template (SfT) methods. They assume that the textured 3D shape-at-rest of the observed object is available as a prior, which is then used to recover the deformation, thus significantly simplifying the problem. This textured 3D shape-at-rest of the object is called the *template*. These methods depend on the deformation model of the template. On one hand, there is the analytic isometric deformation model which assumes that the geodesic distance between points in the surface is preserved. This has proven to be well-posed and quickly evolved to stable and real-time SfT solutions [227, 59, 198]. On the other hand, there are energy-based methods [963, 21, 801, 617, 422], which

*Shape from Template*

jointly minimize the deformation energy with respect to the shape-at-rest and the reprojection error for the image correspondences. These methods have further been embedded within sequential data association with robust kernels to detect and reject outliers [617, 618, 422].

**Non-Rigid Structure from Motion (NRSfM).** Early approaches to deal with the fully-fledged NRSfM problem without the strong prior of SfT assumed an orthographic camera, which is only valid for a moving object far from the camera. These earliest methods were proposed in [113]. This seminal work gave rise to methods based on a low dimensional basis to compute the 3D trajectories from the frames of a sequence [834, 772, 239]. They include regularizers based on either spatial [239, 368], temporal [23], spatio-temporal [20, 394, 395], or —more recently—topological priors [986].

A perspective camera model is a must for the close-up sequences that are typical in deformable SLAM, such as in medical endoscopy. The isometry assumption, first proposed in SfT methods, has also produced excellent results in NRSfM [1073, 1124, 196, 197, 840, 841]. Particularly useful for SLAM is the isometric [840], a local method that is able to naturally handle occlusions and missing data prevalent in deformable SLAM applications. In the transition between SfT and NRSfM, Agudo et al. [22] propose a deformation model based on Navier’s equations and a FEM model processing the video frames in an EKF-SLAM sequential manner.

#### 15.4.1.2 Deformable SLAM

*non-rigid scene models*

In contrast, deformable SLAM targets a variation of the above-mentioned problems, where the scene is deforming *and* the camera is moving. The goal is then to estimate both the history of deformations of the observed scene and the camera trajectory. The hypothesis is that all of the components of the observed deformation that can be explained by a rigid transformation are included in the camera trajectory (see Figure 15.14(b)). The disentanglement of the rigid and non-rigid components of the motion is an ill-posed problem, as highlighted by the so-called ‘Floating Map Ambiguity’ [619]. However, this separation can be achieved by introducing priors on the displacements of the deformable object.

*online vs. offline*

An important SLAM challenge is the fact that the camera is typically undergoing an *exploratory trajectory*, meaning not all the frames observe the same scene region (see Figure 15.14(c)). It is also frequent that the camera is close to the deforming surface, hence the orthographic camera assumption is no longer valid. Figure 15.15 displays this typical close-up in a sequence observing a deforming mandala [618].

The final prevalent assumption is that the process has to be causal and in real-time, thus to produce the map and camera pose at time  $k$ , only the frames from 1 to  $k$  can be processed. This is in sharp contrast to NRSfM where processing of all images is typically performed in batch mode.

Given the difficulties involved, we will first talk about deformable SLAM with

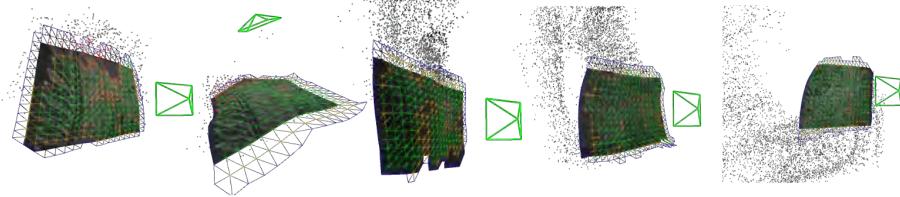


Figure 15.15 Typical exploratory trajectory in Deformable SLAM [618]. The camera pose (green frustum) is estimated together with a local deformable template (mesh) of the deforming scene. Global map points are shown in black. (©2021 IEEE)

depth information in Section 15.4.2 and then come back to deformable SLAM using a monocular camera in Section 15.4.3.

#### 15.4.2 Deformable SLAM with Depth Information

3D sensors such as RGB-D or stereo cameras provide depth measurements. This makes the estimation over-constrained, which greatly reduces the complexity of the problem compared to the pure monocular approach. DynamicFusion [799] is the seminal work on deformable 3D SLAM from RGB-D data. It builds a canonical map of the scene, *i.e.*, its shape at rest, and deforms it in order to explain the current depth observation, combining a form of Embedded Deformation graph model (ED) [1046] with an as-rigid-as-possible regularizer [1022]. ED models build a discretization of the deformations space in a graph structure, speeding up dense reconstructions. DynamicFusion defines the foundations for subsequent works like VolumeDeform [504], that included photometric information to improve the results. Later, KillingFusion [1013] proposed to enforce deformations to be smooth and nearly isometric. An example of this is shown in Figure 15.16. All of the methods above represent the map as SDFs, which scale poorly with the size of the map and limit the application of these methods for exploration. Surfelwarp [364] proposes a surfel representation to be used instead of the classical SDF to improve scalability. In the medical arena, MIS-SLAM [1020] presented deformable SLAM for a stereo system based on as-rigid-as-possible deformations combined with the ED model. Also in stereo medical applications, Zhou and Jayender [1291] propose the Expectation Maximization and Dual Quaternion (EMDQ) algorithm combined with SURF features to track the camera motion and estimate tissue deformation between video frames.

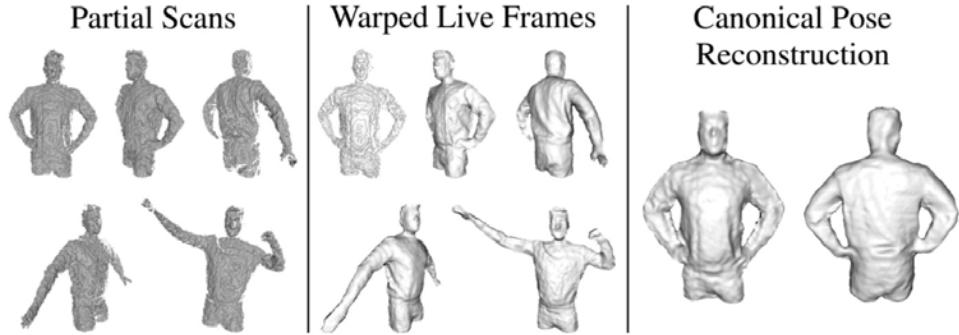


Figure 15.16 KillingFusion [1013] is a method that recovers a non-rigidly moving object observed in a moving RGB-D camera. By using Signed Distance Function (SDF) and a killing regularizer, it is able to recover structures that can undergo topological changes such as the merging of the hands with the torso. (©2017 IEEE)

#### 15.4.3 Pipeline for Deformable SLAM using Monocular Cameras

The typical pipeline in a deformable SLAM system mimics the tracking at frame rate and the mapping at keyframe rate of visual SLAM, but adapted to the deforming case, see, *e.g.*, DefSLAM [618].

**Tracking.** Deformable tracking is similar to SfT methods. The shape-at-rest is assumed available for the local area that is being explored, as firstly proposed in [617], with a deformation energy inspired by physics-based elastic model. Later, Gómez Rodríguez et al. [422] propose a deformable tracking system based on spatial, as-rigid-as-possible, and viscous temporal regularizers. These methods ended up integrated into full deformable SLAM systems in DefSLAM [618] and in NR-SLAM [930], respectively.

**Mapping.** The deformable mapping thread takes point tracks along the sequence as input and is in charge of recomputing the templates at keyframe rate. DefSLAM [618], the first example of deformable SLAM system, uses isometric NRSfM [840] to perform this computation. NRSfM processes a subset of covisible keyframes to produce a template per map keyframe. These templates are aligned with the previous templates computed at earlier stages by previous local mapping operations, assuming a planar topology for the scene. To relax the assumption of planar topology, NR-SLAM [930] introduces the concept of the *dynamic deformable graph* to cope with any scene topology. Regarding the deformation model, the authors go one step further than [422] and propose a intuitive visco-elastic deformation model, which includes both temporal and spatial regularization. This model is brought into the local-mapping deformable BA which is at the core of the deformable mapping, replacing the isometric NRSfM of DefSLAM.

**Feature Matching.** Regarding feature matching, DefSLAM inherits the ORB [950] features from ORB-SLAM [784]. However, they prove to be inadequate descriptors for close-frame tracking in medical scenes, where Lukas-Kanade (LK) optical flow [704] shows better performance [391] due to its invariance to affine lighting changes. In addition, the authors exploit the ORB descriptor combined with a DBoW bag of words [362] for place recognition to achieve camera relocalization after tracking losses. LK tracking is also the method for feature tracking in NR-SLAM [930].

#### 15.4.4 Initialization and Map Extension

Initialization from scratch is always a challenging step for any monocular visual SLAM system. The challenge is even more pronounced in the deformable case. DefSLAM [618, 391] relies on the isometric NRSfM algorithm [840], that is able to compute the templates for the first keyframes from scratch. The only issue is how to compute the initial matches between the keyframes, which is addressed by assuming a planar scene. In the case of NR-SLAM [930], an initial guess for the deformable BA is needed, where the initial map guess and camera motion are computed using rigid SfM. The dynamic deformation graph is initialized from a segmentation of the image optical flow between the initial keyframes.

When the camera explores a new region, new points have to be added to the map to expand the map to cover the new areas. In the case of DefSLAM, the isometric NRSfM naturally handles this extension provided that matches in the keyframes can be found for these new points. In the case of NR-SLAM, the fact that camera poses are available is exploited to triangulate the new map points from their matches in the already located frames. To this end, a model selection approach is proposed where map points are triangulated with both a rigid and a non-rigid algorithm, and then the right model for the newly explored region is selected.

### 15.5 Further Readings & Recent Trends

SLAM in dynamic, changing, and deforming scenes is a highly active and developing area of research. In this chapter, we aimed to provide an overview of key challenges and solution approaches in the field. However, we inevitably could not cover all work that is ongoing, and we hope this serves as inspiration to read more deeply into some of the topics and contribute to the field in the future. In spite of notable advances, many open problems remain. We highlight some of them and other promising future directions below.

**State Estimation in Extreme Environments.** Modern SLAM systems are already quite robust to changes and dynamics in the environment, even without explicitly modeling them. Nonetheless, integrating dynamic object tracking for state

estimation has shown to improve robustness and accuracy if there are large occlusions [688, 450], large fractions of moving objects [894], and strong motion priors [450, 76]. However, these advantages come at increased computational cost and currently each of these challenges is addressed separately. SLAM solutions that are efficient and robust to a diverse range of extreme conditions, or ‘corner cases’, remain an open problem.

**Differentiable Scene Representations.** Differentiable rendering models such as NeRF and 3DGS as introduced in Chapter 14, although still recent, have already shown a lot of promise as representation for dynamic scenes. Initial works demonstrate their capacity for high-fidelity reconstruction of dynamic and deforming objects [1189] as well as for change detection [703]. Nonetheless, this is still a developing field with open challenges in memory and computation cost for real-time SLAM, partial and noisy observations from mobile sensors, as well as scaling and map management for spatially and temporally consistent mapping.

**Data Association and Deep Learned Priors.** The fundamental challenge of data association is still a hard and relevant problem, especially for partial observations, occlusions, as well as long-term place recognition and instance re-localization. To overcome this, there appears to be a trend toward deep learned methods that incorporate priors to complete and associate objects from partial observations [345, 1304, 1204]. In particular, geometric deep learning techniques to extract SE(3)-invariant or SE(3)-equivariant features hold promise to address these challenges. However, there are still open questions with respect to generalization, reliability, and scalability for mobile deployment.

**Lifelong Operation and Continual Learning.** A further open problem is the scalability and performance of SLAM methods for lifelong operation. This includes a range of perceptual challenges from appearance to geometric changes, as well as possible sensor degradation over time. Furthermore, if a robot can operate for long durations, it should also be able to continually learn [1131] to adapt and improve over time. Finally, both of these directions will need some form of memory management to decide which information to retain, how to summarize or marginalize, and when and what to forget.

**Deformable SLAM.** The research on SLAM in deformable environments is still at its earlier stage. There are still many open research questions [492]. For example, under what conditions is it possible to accurately estimate both the camera trajectory and the deformation of the map? Can we clearly distinguish the rigid motion of the deformable object and the rigid motion of the camera? Can we provide accurate estimates of the uncertainties of the deformable SLAM results? Are there local minima in deformable SLAM? How can we know whether the obtained result is the global minimum or not?

**Towards Spatio-Temporal AI.** Beyond SLAM in dynamic and deformable scenes, spatio-temporal AI aims to build a holistic scene understanding that goes beyond localization and reconstruction, but captures and learns temporal patterns.

Ideally, such a map or ‘world model’ will combine semantic priors and observations from the robot to provide a profound and mechanistic understanding of a scene and its dynamics. This will allow estimating a consistent and complete history of the past, predicting future outcomes, and facilitate high-level and long-term reasoning and decision-making.

# 16

## Metric-Semantic SLAM

Arash Asgharivaskasi, Kevin Doherty, Jens Behley, Nathan Hughes, Yun Chang,  
John Leonard, Henrik I. Christensen, Luca Carlone, and Nikolay Atanasov

This chapter discusses how geometric map representations produced by SLAM methods can be augmented into more semantically rich representations to enable a broader set of downstream tasks, while also enhancing the SLAM performance. SLAM plays a critical role in mobile robot autonomy, by enabling robots to localize themselves and maintain a consistent map representation of a priori unknown environments. As we discussed in Chapter I, another equally important role of SLAM is to support robot task and motion planning by providing information about task and motion goals and constraints. Dense map representations (as the ones we reviewed in Chapter 5) offer efficient ways to encode safety constraints for collision checking in robot navigation and manipulation applications. Examples include motion planning and trajectory optimization supported by occupancy [94, 170], signed distance function [819, 1274], and mesh [885, 112] representations. However, as one considers encoding more complex robot tasks, including semantic goals and requirements (*e.g.*, “navigate to the laptop in the office” or “avoid entering the meeting room if there are people inside”), geometric information about the environment alone may be insufficient. In the context of robot navigation, information about the semantics of the environment, *e.g.*, whether a portion of the map is a road, sidewalk, or vegetation, can be utilized to specify different traversability costs [907, 663]. More generally, map representations generated by SLAM techniques can support robot task specification and planning by providing semantic information about objects, places, and their relations. Such information is not only useful for downstream tasks, but is often useful for the SLAM system itself: semantically meaningful features are more uniquely identifiable and viewpoint invariant, leading to improvements in data association and loop closure [108, 707]. Similarly, semantic information can help handle dynamic entities in the scene [940], which are typically less informative for motion estimation (*cf.* Chapter 15).

Being able to construct semantically rich map representations requires extracting additional information from the visual observations beyond image-gradient-based keypoints. Advances in deep learning allow obtaining object detections, object keypoints, semantic edges, semantic segmentation, instance segmentation, panoptic segmentation, or other semantic information from the visual observations. *Object*

*detection* is a computer vision task that identifies and localizes objects within an image, typically via bounding box annotations. *Semantic keypoints* [851] or *semantic edges* [1254] can be identified as mid-level parts of objects (*e.g.*, doors, wheels, windshield of a car). *Semantic segmentation* [444, 931] goes beyond object detection by assigning a class label to every pixel in an image. *Instance segmentation* also identifies and separates individual objects, even if they belong to the same category, by creating pixel-by-pixel masks around each object instance. *Panoptic segmentation* [579] combines semantic and instance segmentation —by segmenting both the background and object instances— to achieve a comprehensive understanding of an image.

This chapter describes different extensions of traditional SLAM to utilize semantic information from the visual front-end and to generalize the map optimized by the back-end to accumulate and fuse this information in a 3D spatial metric-semantic representation. In particular, we start with a quick overview in Section 16.1. Then, Section 16.2 delves into how to augment landmark-based SLAM with information about the landmark semantics. Section 16.3 describes extensions of the dense representations in Chapter 5 to incorporate semantic information. Section 16.4 discusses hierarchical map representations, that capture semantics at different levels of abstraction. As usual, we close the chapter with a discussion about recent trends in Section 16.5. We remark that this chapter focuses on “closed-set” semantics, *i.e.*, the case where semantic information is restricted to a relatively small (*e.g.*, 100-1000 labels) and predefined dictionary of semantic concepts (*e.g.*, ‘chair’, ‘table’, ‘office’). We postpone discussing how to incorporate open-set semantics (*i.e.*, language embeddings) into SLAM to Chapter 17.

## 16.1 From Traditional SLAM to Metric-Semantic SLAM

There are multiple ways to include semantic information in a map representation. For instance, one can include sparse semantic information by adding semantics to the landmarks in landmark-based maps, embed semantic labels in dense surface-based maps, or even assign semantics to entire regions of free space (*e.g.*, rooms in indoors, or more general regions, such as a parking lot, in outdoors). In the case of sparse maps, this involves a generalization of the usual 3D point landmarks to represent objects with their categories, and possibly also include their poses and shapes. The associated sensor measurement models also need to be extended from low-level visual keypoints and features to high-level detections such as bounding boxes [917], segmentation masks [444], or object parts [851]. This, in turn, leads to *novel formulations of probabilistic factors* and novel factor-graph optimization techniques that capture both metric and semantic information. In the case of dense maps, semantic information can be captured by extending occupancy models from a binary representation of free and occupied space to a *multi-class* representation of semantic categories. More generally, semantic features extracted by computer

vision models at the front-end can be grounded to points, surfels, mesh faces, or voxels maintained in dense maps. Besides a transformation of semantic observations to 3D space, this requires the formulation of novel observation models for semantic information and sequential probabilistic inference techniques for semantic map information updates. As we will see in Section 16.4, hierarchical mapping approaches often combine sparse and dense metric-semantic maps into a single unified representation, describing semantics at multiple levels of abstraction, from dense surface-based maps, to objects and regions.

## 16.2 Sparse Metric-Semantic Representations

Consider a landmark-based SLAM problem in which the landmarks represent physical objects in the environment. We refer to this setting as *sparse metric-semantic SLAM* because the map consists of object landmarks with metric properties, such as position, orientation, and shape, and semantic properties, such as object category.

As discussed in Chapter I, landmark-based SLAM is often formulated as a nonlinear least-squares problem, where the landmark observations induce squared terms  $r_i(\mathbf{x}_i)^2$  in the objective function, and residual errors have the following form:

$$r_i(\mathbf{x}_i) = \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{T}_i, \boldsymbol{\ell}_i)\|_{\Sigma_i}, \quad (16.1)$$

with measurement  $\mathbf{z}_i$  and state variable  $\mathbf{x}_i = (\mathbf{T}_i, \boldsymbol{\ell}_i)$  containing robot pose  $\mathbf{T}_i$  and landmark state  $\boldsymbol{\ell}_i$ . Note that similar to earlier chapters, the index  $i$  refers to the factor introduced in the factor graph by the paired measurement  $\mathbf{z}_i$  and state  $\mathbf{x}_i$  assuming known data association. In visual SLAM (Chapter 7), the landmarks  $\boldsymbol{\ell}_i$  are most commonly represented as 3D points, and the associated measurements  $\mathbf{z}_i$  are 2D pixel coordinates of the landmark projection onto the image plane. In sparse metric-semantic SLAM, the landmarks are generalized to model objects and their properties. Measurements and residual errors are also generalized to model object detections from computer vision algorithms, and measure error with respect to projections of objects from 3D space to the 2D image plane. Object representations, object measurements, and object residual error definitions are discussed next.

### 16.2.1 Object Representation and Factor Graph Modeling

We begin by extending the notion of landmark from a 3D point to a rigid-body object. An object landmark is characterized by not only its position but also its orientation, scale, shape, and semantic category.

*object landmark*

**Definition 16.1** An *object landmark* is a tuple  $\boldsymbol{\ell} = (\sigma, \mathbf{q}, \lambda, \mathbf{s})$  including a semantic category  $\sigma \in \mathbb{N}$  (e.g., “car”, “chair”, “table”), pose  $\mathbf{q} \in \text{SE}(3)$ , scale  $\lambda \in \mathbb{R}_{>0}$ , and shape  $\mathbf{s} \in \mathbb{R}^d$ .

The pose  $\mathbf{q}$  of an object landmark  $\ell$  specifies the position and orientation of the object coordinate frame with respect to the world coordinate frame of the map. In addition to a rigid-body transformation, the definition of an object local coordinate frame requires scaling to allow the presence of objects from the same category and shape but of different sizes (*e.g.*, a car vs. a toy car). We present possible descriptions of object shape next and illustrate object landmark representations in several examples. Similar to dense surface modeling in SLAM (Chapter 5), the shape  $\mathbf{s}$  of an object can be represented using an *explicit surface model* (*e.g.*, points, elementary geometric shapes, mesh) or an *implicit surface model* (*e.g.*, occupancy function, signed distance function field). We present commonly used object shape representations in object SLAM below.

#### 16.2.1.1 Object Shape Representation as Semantic Landmarks

The shape of an object from a given category (Definition 16.1) can be modeled as a collection  $\{\mathbf{s}_j\}_j$  of sparse 3D points  $\mathbf{s}_j \in \mathbb{R}^3$  called *semantic landmarks*. Typically, semantic landmarks correspond to distinctive parts of an object; for instance, for a car, we might attach semantic keypoints to the wheels, mirrors, plate, etc. While this shape model may be viewed simply as a point cloud, it is necessary for the SLAM backend optimization to relate the 3D semantic landmarks to the 2D visual observations and measure the corresponding residual error. Therefore, instead of a generic point cloud, the semantic landmarks  $\mathbf{s}$  are defined as keypoints located in a 3D Computer Aided Design (CAD) object model using a deformable (or active) shape model [231]:

$$\mathbf{s}_j = \mathbf{b}_{j,0} + \sum_k c_k \mathbf{b}_{j,k}, \quad (16.2)$$

where  $\mathbf{b}_{j,0}$  are semantic landmarks from an average category-level shape model and  $\mathbf{b}_{j,k}$  are several modes of shape variability obtained by PCA with associated shape deformation coefficients  $c_k \in \mathbb{R}_{\geq 0}$  as described in [851]. For example, the average shape of a car object may be represented by semantic landmarks  $\mathbf{b}_{j,0}$  but, to allow shape variations in car instances of different makes and models, we allow shape deformation along the principal modes of variability  $\mathbf{b}_{j,k}$  with coefficients  $c_k$ . Hence, the shape of an observed object landmark may be estimated by optimizing  $\{\mathbf{s}_j\}_j$  directly or by optimizing the shape deformation coefficients  $c_k$  [1003]. Contrary to optimizing independent points, the parametrization (16.2) enforces the semantic landmarks to describe plausible objects shapes.

We refer to the projection of a semantic landmark  $\mathbf{s}_j$  onto the image plane of a camera as a *semantic keypoint*  $\mathbf{z}_j \in \mathbb{R}^2$ . Semantic keypoints can be detected in camera images using convolutional neural network models trained using supervised learning to provide heatmaps consisting of 2D Gaussians centered at each keypoint [851, 1297]. This is illustrated in Figure 16.1.

The residual error between an object landmark  $\ell_i$ , with world-frame pose  $\mathbf{q}_i$ , scale

*semantic landmark*

*semantic keypoint*



Figure 16.1 Semantic keypoints extracted from an RGB measurement. (a) An outdoor environment with two cars and a robot equipped with a camera. (b) RGB measurement in the camera frame. (c) Semantic keypoints detected in the RGB image.

$\lambda_i$ , and semantic landmark shape  $\{\mathbf{s}_{i,j}\}_j$ , and corresponding semantic keypoint detections  $\{\mathbf{z}_{i,j}\}_j$  obtained from camera pose  $\mathbf{T}_i$  (transformation from camera optical frame to world frame) is measured using the camera perspective projection model:

$$r_i(\mathbf{x}_i)^2 = \sum_j \|\mathbf{z}_{i,j} - \boldsymbol{\pi}(\mathbf{q}_i^{-1}\mathbf{T}_i, \lambda_i \mathbf{s}_{i,j})\|_{\Sigma_i}^2, \quad (16.3)$$

where  $\boldsymbol{\pi}(\mathbf{T}, \mathbf{s}_j)$  denotes the perspective projection of 3D point  $\mathbf{s}_j$  to 2D pixel coordinates in the image plane of a camera with pose  $\mathbf{T}$ . In detail, the camera pose in the object frame is  $\mathbf{q}_i^{-1}\mathbf{T}_i$ , and each semantic landmark  $\mathbf{s}_{i,j}$  (with object-frame coordinates) is first scaled by  $\lambda_i$  and then transformed from the object coordinate frame to camera coordinate frame. The predicted pixel coordinates of the object semantic landmarks are compared to the semantic keypoint detections  $\mathbf{z}_{i,j}$  by the residual error in (16.3).

An example of object SLAM using semantic landmarks as the object shape representation applied to the real-world KITTI dataset [373] is shown in Figure 16.2. The approach is an extension of that in [108] to utilize multiple semantic landmarks per object instead of just the object centroid.

#### 16.2.1.2 Object Shape Representation as Quadric Surface

Another commonly used explicit model of object shape is based on quadric surfaces [802, 818, 992]. In this case, we model the shape of an object landmark as an ellipsoid  $\mathcal{E}_s \subset \mathbb{R}^3$  with semi-axis lengths  $\mathbf{s} \in \mathbb{R}^3$  defined as:

$$\mathcal{E}_s = \{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{x}^\top \text{diag}(\mathbf{s})^{-2} \mathbf{x} \leq 1\}. \quad (16.4)$$

*quadric surface*

An ellipsoid is an example of a *quadric surface* [802] defined by matrix  $\mathbf{Q}_s \in \mathbb{R}^{4 \times 4}$

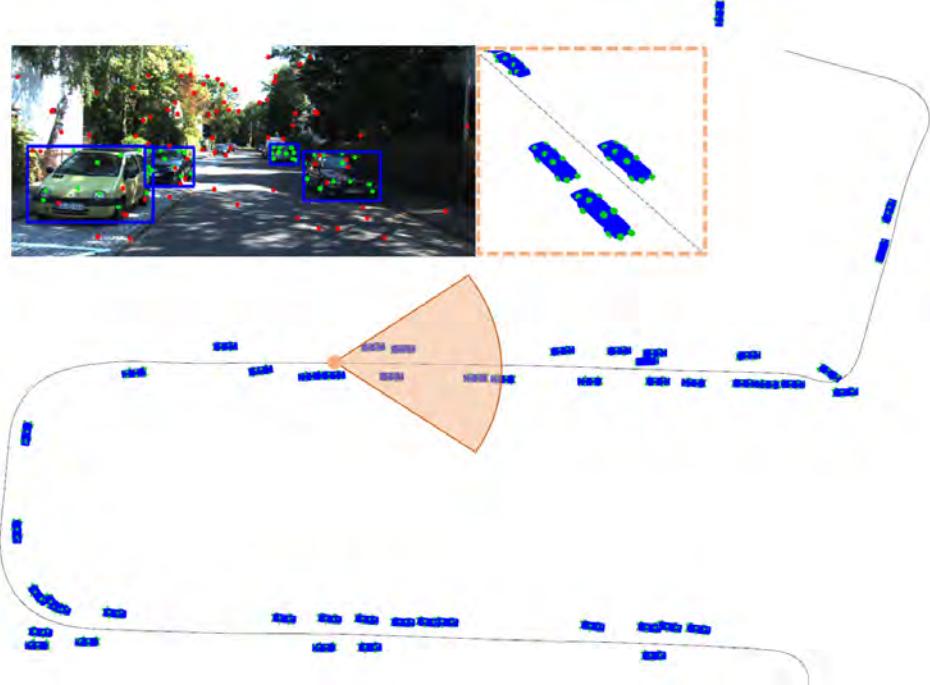


Figure 16.2 Object detections (top left blue bounding boxes), semantic keypoints (top left green 2D points), and regular visual keypoints (top left red 2D points) extracted from RGB images in the KITTI dataset [373] are used to estimate car semantic landmarks (green 3D points, *e.g.*, wheels, front lights), car poses (visualized as blue meshes), and the sensor trajectory (black curve).

obtained by rewriting the inequality in (16.4) in homogeneous coordinates:

$$\mathcal{E}_s = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid \underbrace{\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^\top \begin{bmatrix} \text{diag}(\mathbf{s})^{-2} & \mathbf{0} \\ \mathbf{0}^\top & -1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}}_Q \leq 0 \right\}. \quad (16.5)$$

An ellipsoid representation of object shape is attractive because its projection to the image of a camera can be obtained analytically and, in turn, can be compared to bounding-box object detections using a residual error term (see illustration in Figure 16.3). The relationship between a 3D ellipsoid and its perspective projection to a 2D conic can be obtained analytically using a dual quadric representation. The dual of an ellipsoid  $\mathcal{E}_s$  is the set of all hyperplanes tangent to it:

$$\mathcal{E}_s^* = \{ \mathbf{y} \in \mathbb{R}^3 \mid \mathbf{y}^\top \text{diag}(\mathbf{s})^2 \mathbf{y} \leq 1 \} = \left\{ \mathbf{y} \in \mathbb{R}^3 \mid \underbrace{\begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix}^\top Q_s^* \begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix}}_{Q} \leq 0 \right\}, \quad (16.6)$$

where  $Q_s^*$  is the adjugate of  $Q_s$ . For an ellipsoid,  $Q_s$  is invertible and  $Q_s^* =$

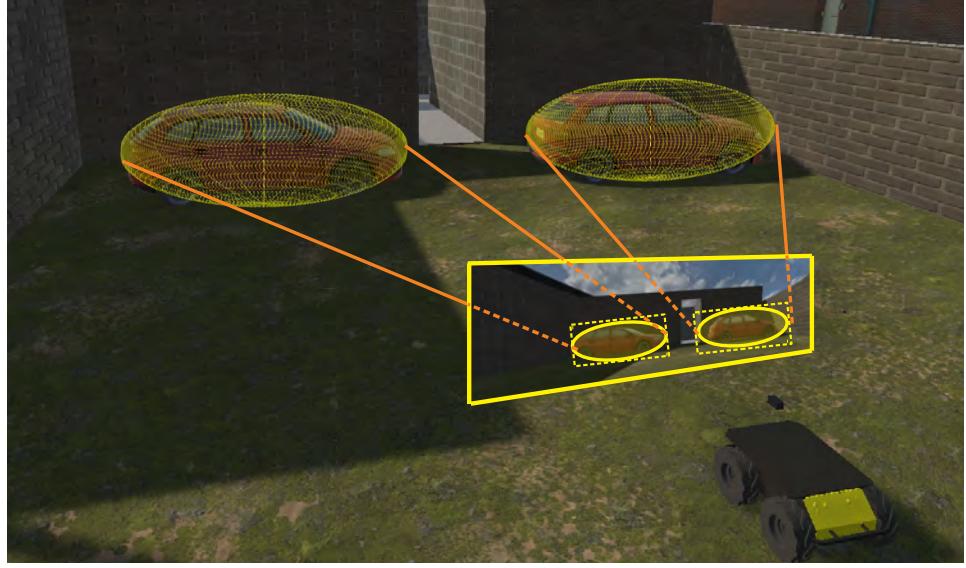


Figure 16.3 Ellipsoid representation of detected objects. The 2D projection of the 3D ellipsoids onto the robot camera frame is shown, in addition to the detection bounding boxes.

$\text{adj}(\mathbf{Q}_s) = \det(\mathbf{Q}_s)\mathbf{Q}_s^{-1}$ , which can be simplified to  $\mathbf{Q}_s^{-1}$  due to the scale invariance of the dual quadric definition in (16.6).

Consider an object landmark  $\ell$  with pose  $\mathbf{q}$ , scale  $\lambda$ , and shape  $s$  describing a quadric surface  $\mathbf{Q}_s$ . The object surface is scaled and transformed from the local object coordinate frame to the world coordinate frame as follows:

$$\mathbf{Q}_\ell^* \propto \mathbf{q} \begin{bmatrix} \lambda \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{Q}_s^* \begin{bmatrix} \lambda \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}^\top \mathbf{q}^\top. \quad (16.7)$$

The perspective projection of the dual quadric  $\mathbf{Q}_\ell^*$  from the world coordinate frame to the image plane of a camera with pose  $\mathbf{T} \in \text{SE}(3)$  is a dual conic  $\mathbf{C}_z^* \in \mathbb{R}^{3 \times 3}$  defined by:

$$\mathbf{C}_x^* \propto \frac{1}{\beta} \mathbf{P} \mathbf{T}^{-1} \mathbf{Q}_\ell^* \mathbf{T}^{-\top} \mathbf{P}^\top, \quad (16.8)$$

where  $\mathbf{P} = [\mathbf{I}_3 \quad \mathbf{0}] \in \mathbb{R}^{3 \times 4}$  is a projection matrix that drops the last coordinate and  $\beta$  captures the depth scaling of the camera. The subscript  $x$  emphasizes that  $\mathbf{C}_x^*$  is determined by the state variable  $\mathbf{x} = (\mathbf{T}, \ell)$ , including both the camera pose  $\mathbf{T}$  and the object landmark  $\ell$ .

*bounding box*

An object *bounding-box detection*  $\mathbf{z} = [u \ v \ w \ h]^\top \in \mathbb{R}^4$  consists of the normalized pixel coordinates  $(u, v)$  of the bottom-left bounding-box corner (after application of the inverse camera intrinsics matrix) and the width and height  $(w, h)$

of the bounding box. The dual conic representation of an axis-aligned ellipse fitting inside the bounding box  $\mathbf{z}$  with center  $(c_u, c_v) = (u + w/2, v + h/2) \in \mathbb{R}^2$  is:

$$\mathbf{C}_{\mathbf{z}}^* \propto \begin{bmatrix} 1 & 0 & c_u \\ 0 & 1 & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (w/2)^2 & 0 & 0 \\ 0 & (h/2)^2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & c_u \\ 0 & 1 & c_v \\ 0 & 0 & 1 \end{bmatrix}^\top \quad (16.9)$$

Thus, the residual error between an object landmark  $\ell_i$ , with pose  $\mathbf{q}_i$ , scale  $\lambda_i$ , and quadric shape  $\mathbf{Q}_{\mathbf{s}_i}$ , and corresponding bounding box detection  $\mathbf{z}_j$  obtained from camera pose  $\mathbf{T}_i$  can be measured by the discrepancy between the conic  $\mathbf{C}_{\mathbf{z}_j}^*$  in (16.9) obtained from the bounding box and the conic  $\mathbf{C}_{\mathbf{x}_i}^*$  in (16.8) obtained by projecting the quadric shape to the image plane:

$$r_i(\mathbf{x}_i) = \|\mathbf{C}_{\mathbf{z}_j}^* - \mathbf{C}_{\mathbf{x}_i}^*\|_{\Sigma_i}. \quad (16.10)$$

To avoid defining a matrix norm and exploit the fact that conics are defined by symmetric matrices, we can vectorize  $\mathbf{C}_{\mathbf{z}_j}^* - \frac{1}{\beta_i} \mathbf{P} \mathbf{T}_i^{-1} \mathbf{Q}_{\ell_i}^* \mathbf{T}_i^{-\top} \mathbf{P}^\top$ . In detail, define  $\mathbf{b}_j = \text{vech}(\mathbf{C}_{\mathbf{z}_j}^*)$  as the half-vectorization of the symmetric matrix  $\mathbf{C}_{\mathbf{z}_j}^*$  and similarly define  $\mathbf{v}(\mathbf{T}_i, \ell_i) = \text{vech}(\mathbf{T}_i^{-1} \mathbf{Q}_{\ell_i}^* \mathbf{T}_i^{-\top})$ . Let  $\mathbf{A} = \mathbf{D}(\mathbf{P} \otimes \mathbf{P})\mathbf{E}$ , where  $\otimes$  is the Kronecker product and matrices  $\mathbf{D} \in \mathbb{R}^{6 \times 9}$  and  $\mathbf{E} \in \mathbb{R}^{16 \times 10}$  are such that  $\text{vech}(\mathbf{Q}) = \mathbf{D} \text{vec}(\mathbf{Q})$  and  $\text{vec}(\mathbf{Q}) = \mathbf{E} \text{vech}(\mathbf{Q})$ . Thus, we can rewrite the residual error in (16.10) as:

$$r_i(\mathbf{x}_i) = \|\mathbf{b}_j - \frac{1}{\beta_i} \mathbf{A} \mathbf{v}(\mathbf{T}_i, \ell_i)\|_{\Sigma_i}. \quad (16.11)$$

The reader may refer to [949] for details and to [802, 818] for alternative residual error formulations.

An example of object SLAM with the OrcVIO algorithm [992] —combining semantic landmarks and ellipsoids as the object shape representation— applied to real-world data is shown in Figure 16.4.

#### 16.2.1.3 Object Shape Representation as Mesh

A more expressive object shape representation can be obtained by using a triangular *mesh* with vertices  $\mathcal{V} = \{\mathbf{s}_j\}_j$  with 3D coordinates  $\mathbf{s}_j \in \mathbb{R}^3$  and faces  $\mathcal{F} \subset \mathcal{V} \times \mathcal{V} \times \mathcal{V}$ . *mesh*  
To estimate the parameters of an object landmark with mesh shape, we associate a rendering of the mesh with a *segmentation mask*  $\mathbf{z}_i \subset \mathbb{R}^2$  in a camera image obtained from detecting and segmenting the object [444, 917, 1118]. *segmentation mask* This is illustrated in Figure 16.5.

To optimize the mesh shape of an object in SLAM, we need to define a differentiable residual error. The key challenge is to project and rasterize the mesh in the image plane using differentiable rendering. Given a mesh with vertices  $\mathcal{V}$  and faces  $\mathcal{F}$ , a rasterization function,  $\rho(\mathcal{V}, \mathcal{F})$ , can be defined by projecting the mesh vertices to the image plane and drawing only the front-most face at each pixel when multiple faces are present [730]. Kato et al. [540] and Liu et al. [677] were among the first to obtain an approximate gradient for the rasterization function  $\rho$  with

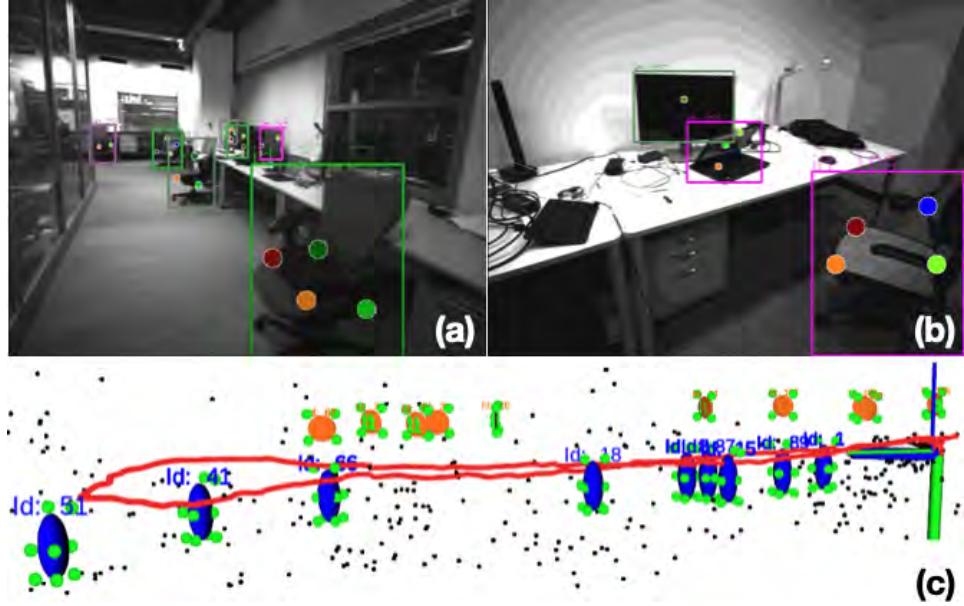


Figure 16.4 Localization and object mapping in an indoor scene with chairs and monitors. Bounding-box and semantic-keypoint detections are shown in (a) and (b). The estimated sensor trajectory (red curve), geometric landmarks (black dots), semantic landmarks (green dots), and object ellipsoids (blue for chairs, orange for monitors) obtained by OrcVIO [992] are shown in (c).

respect to the mesh vertices. The reader is invited to refer to [541] for a survey on differentiable rendering.

The residual error between an object landmark  $\ell_i$ , with pose  $\mathbf{q}_i$ , scale  $\lambda_i$ , and mesh shape  $(\mathcal{V}, \mathcal{F})$ , and the corresponding segmentation mask  $\mathbf{z}_i$  obtained from robot pose  $\mathbf{T}_i$ , is measured using the reciprocal of the intersection-over-union between the mesh image-plane projection and the segmentation mask:

$$r_i(\mathbf{x}_i) = \frac{\rho(\{\pi(\mathbf{q}_i^{-1}\mathbf{T}_i, \lambda_i \mathbf{s}_{i,j})\}_j, \mathcal{F}) \cup \mathbf{z}_i}{\rho(\{\pi(\mathbf{q}_i^{-1}\mathbf{T}_i, \lambda_i \mathbf{s}_{i,j})\}_j, \mathcal{F}) \cap \mathbf{z}_i}, \quad (16.12)$$

where  $\pi(\mathbf{q}_i^{-1}\mathbf{T}_i, \lambda_i \mathbf{s}_{i,j})$  is the perspective projection of the 3D mesh vertices  $\mathbf{s}_{i,j}$  to the image plane as in (16.3) and  $\rho$  is a differentiable mesh rasterization function. Similar to the semantic landmark representation above, the mesh vertex coordinates  $\{\mathbf{s}_j\}_j$  can be optimized directly or indirectly by defining an average category-level shape and optimizing the deformation coefficients of the principle components of shape variability. Other residual error function  $r_i$  such as the  $\ell_2$  loss or the binary cross entropy between the rendered object mask  $\rho(\{\pi(\mathbf{q}_i^{-1}\mathbf{T}_i, \lambda_i \mathbf{s}_{i,j})\}_j, \mathcal{F})$  and the segmentation mask  $\mathbf{z}_i$  may be used [699].

Differentiable mesh rendering has been used for object SLAM in [323], for human



Figure 16.5 Mesh representation of the shape of detected objects. (a) Bounding boxes and segmentation masks of detected objects in the camera view, highlighted in blue. (b) Result of minimizing the residual error between the segmentation masks and the projections of the object meshes.

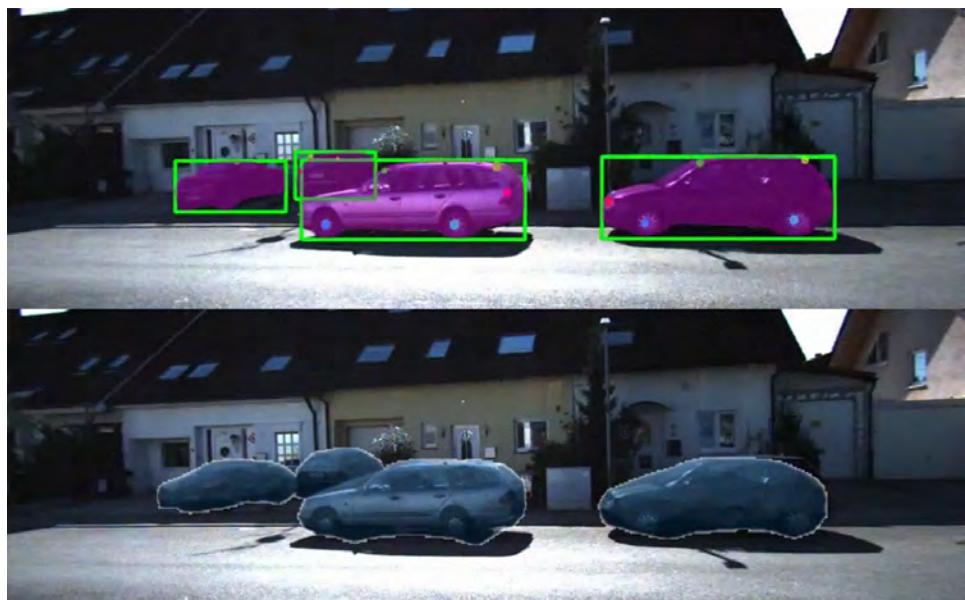


Figure 16.6 Qualitative results of 3D object mesh shape and pose estimation using differentiable segmentation masks [323] on the KITTI odometry dataset [373]. The method takes bounding boxes (green), segmentation masks (magenta) and semantic keypoints (multiple colors) as input and optimizes object poses and mesh shapes using the intersection-over-union residual in (16.12).

pose estimation in [852], and for robot pose estimation in [699]. An example of object SLAM from [323] using meshes as the object shape representation applied to the real-world KITTI dataset [373] is shown in Figure 16.6. A similar example from [1158], generating differentiable segmentation masks from a deformable shape model (16.2) of a voxel grid storing signed distance values is shown in Figure 16.7.

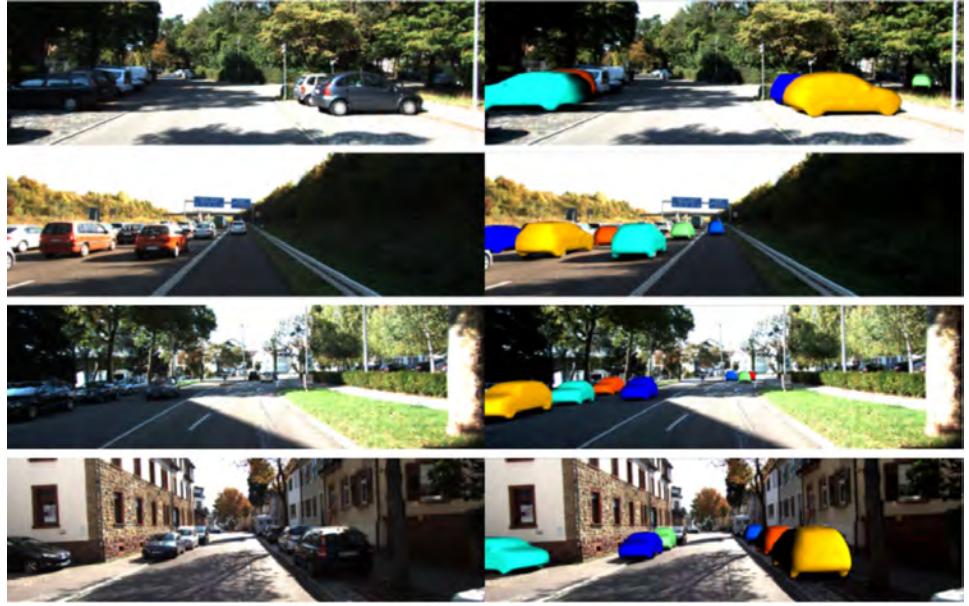


Figure 16.7 Qualitative results of 3D object shape and pose estimation using differentiable segmentation masks obtained from deformable signed distance shape model [1158] on the KITTI Stereo 2015 benchmark [753]. The method takes a stereo image with segmentation masks, initial object pose and learned object mean shape as input. The object is projected to the images and the consistencies between the projections and the segmentation masks are measured by silhouette alignment and photometric consistency residuals to optimize the object pose and shape.

#### 16.2.1.4 Implicit Object Shape Representations

The semantic landmarks, quadric surface, and mesh representations of object shape discussed above are examples of explicit shape models because they represent the object surface geometry directly. Next, we discuss implicit shape representations, which model an object as a spatial function, where a particular level set of this function represents the object’s surface. Widely used implicit shape representations include occupancy functions [756], SDFs [844], and NeRFs [1275].

Consider an object landmark with shape represented as a set  $\mathcal{S} \subset \mathbb{R}^3$ . The *occupancy function*  $f_{\mathcal{S}}(\mathbf{x})$  of set  $\mathcal{S}$  is a binary function indicating whether a point  $\mathbf{x} \in \mathbb{R}^3$  is inside  $\mathcal{S}$ :

$$f_{\mathcal{S}}(\mathbf{x}) = \begin{cases} -1 & \text{if } \mathbf{x} \in \mathcal{S}, \\ 1 & \text{if } \mathbf{x} \notin \mathcal{S}. \end{cases} \quad (16.13)$$

*signed distance function* The *signed distance function*  $d_{\mathcal{S}}(\mathbf{x})$  of set  $\mathcal{S}$  is a real-valued function measuring the

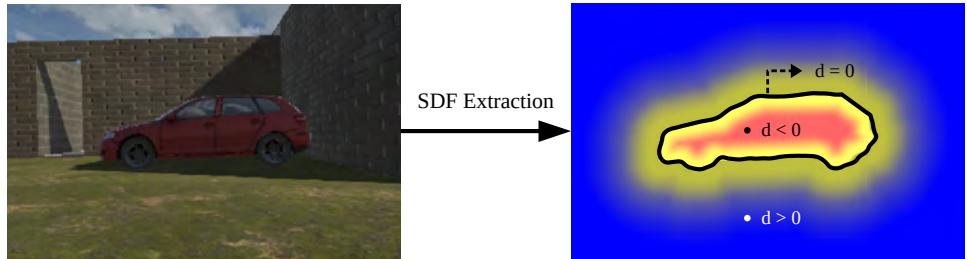


Figure 16.8 SDF of an object landmark. By definition, the object surface is at zero distance, while the distances inside and outside of the object are negative and positive, respectively.

signed distance from a point  $\mathbf{x} \in \mathbb{R}^3$  to the boundary of  $\mathcal{S}$ :

$$d_{\mathcal{S}}(\mathbf{x}) = \begin{cases} -\inf_{\mathbf{y} \in \partial\mathcal{S}} \|\mathbf{x} - \mathbf{y}\|, & \text{if } \mathbf{x} \in \mathcal{S}, \\ \inf_{\mathbf{y} \in \partial\mathcal{S}} \|\mathbf{x} - \mathbf{y}\|, & \text{if } \mathbf{x} \notin \mathcal{S}. \end{cases} \quad (16.14)$$

The SDF definition is illustrated in Figure 16.8.

Given the definitions above, reconstructing an implicit model of object shape from sensor measurements can be viewed as a regression problem to estimate the object’s occupancy function  $f_{\mathcal{S}}(\mathbf{x})$  or SDF  $d_{\mathcal{S}}(\mathbf{x})$ . We focus our discussion on estimating SDF shape but a similar approach can be used to obtain occupancy models instead. To approximate  $d_{\mathcal{S}}(\mathbf{x})$ , we introduce a neural network  $d_{\theta}(\mathbf{x}, \mathbf{s})$  with parameters  $\theta$  and latent feature vector  $\mathbf{s} \in \mathbb{R}^d$  (referred to as *shape code*) that captures the specific shape of  $\mathcal{S}$ . In other words, we can think of  $d_{\theta}$  as a neural network decoder that estimates the signed distance from a query point  $\mathbf{x}$  to the surface of an object with shape code  $\mathbf{s}$ . Varying  $\mathbf{s}$  allows us to consider variations of object shapes with the same neural network decoder, which is useful for category-level shape representation [844, 993].

To estimate the SDF of an object landmark  $\ell_i$ , we consider a distance sensor, such as a LiDAR or depth camera, that provides data  $\{\mathbf{y}_{i,j}, \mathbf{z}_{i,j}\}_j$  from robot pose  $\mathbf{T}_i$  consisting of point measurements  $\mathbf{y}_{i,j}$  near the surface of object  $\ell_i$  and distance measurements  $\mathbf{z}_{i,j}$  to the object surface. Then, the residual error between an object landmark  $\ell_i$ , with pose  $\mathbf{q}_i$ , scale  $\lambda_i$ , and shape code  $\mathbf{s}_i$ , and corresponding observation  $\{\mathbf{y}_{i,j}, \mathbf{z}_{i,j}\}_j$  from robot pose  $\mathbf{T}_i$  can be measured by comparing the SDF prediction to the measured distance:

$$r_i(\mathbf{x}_i)^2 = \sum_j |\mathbf{z}_{i,j} - d_{\theta}(\lambda_i^{-1} \mathbf{q}_i \mathbf{T}_i^{-1} \mathbf{y}_{i,j}, \mathbf{s}_i)|_{\sigma_i}^2, \quad (16.15)$$

where  $\lambda_i^{-1} \mathbf{q}_i \mathbf{T}_i^{-1} \mathbf{y}_{i,j}$  transforms the point  $\mathbf{y}_{i,j}$ , first from the robot frame to the world frame, then from the world frame to the object frame, and finally scales it by  $\lambda_i^{-1}$  to obtain a query point in the canonical object frame [1149]. Assuming

that the SDF decoder is already trained, the residual in (16.15) can be minimized to simultaneously estimate the robot pose  $\mathbf{T}_i$  and the pose  $\mathbf{q}_i$ , scale  $\lambda_i$ , and shape code  $\mathbf{s}_i$  of the object landmark  $\ell_i$ .

SDF regression methods often introduce other residual terms, *e.g.*, relating normals at the object surface to the gradient of  $d_\theta$  via an Eikonal equation [404] or encouraging positive/negative values far away from the surface [831].

We distinguish between the training phase, where we optimize the parameters  $\theta$  of the SDF decoder of an object category using offline data, and the testing phase, where we optimize the pose  $\mathbf{q}_i$ , scale  $\lambda_i$ , and shape code  $\mathbf{s}_i$  of a previously unseen object instance from a category with pretrained decoder using online data. In training, one usually assumes that the sensor pose  $\mathbf{T}_i$  and landmark pose  $\mathbf{q}_i$  and scale  $\lambda_i$  are known, and the residual in (16.15) is minimized with respect to  $\theta$  and  $\mathbf{s}_i$ . The decoder parameters  $\theta$  are usually the same for a whole object category, while the shape code  $\mathbf{s}_i$  is optimized separately for each object instance. In testing, one usually assumes that the decoder parameters  $\theta$  are known, and the residual in (16.15) is minimized with respect to the state  $\mathbf{x}_i$ , consisting of  $\mathbf{T}_i$ ,  $\mathbf{q}_i$ ,  $\lambda_i$ , and  $\mathbf{s}_i$ .

An example of object SLAM with the ELLIPSDF algorithm [993], using an implicit representation of both coarse (ellipsoid) and fine (SDF) object shape, applied to the real-world ScanNet dataset [238] is shown in Figure 16.9.

### 16.2.2 Hybrid Solvers for Sparse Metric-Semantic SLAM

The representation of object-based or *sparse* metric-semantic SLAM combines *discrete* information in the form of semantic object *classes* with *continuous* information, like the position, orientation, and shape of an object. Consequently, solving metric-semantic SLAM problems typically requires reasoning jointly about these states which are often coupled in intricate ways. For example: the semantic class of an object might inform us about the shape of that object (*e.g.*, as in [991]) and vice versa. Discrete variables enter the sparse metric-semantic SLAM problems in other ways, too, such as in the case where objects of interest have some discrete symmetries [702]. When using a learning-based front-end system for object detection and pose estimation, like a neural network, occasionally the front-end will provide multiple discrete pose hypotheses we would like to track (see, *e.g.* [344]). Crucially, much like in the case of landmark-based SLAM with purely geometric features, *data association* (see Chapter 3) is a key challenge when developing systems for sparse metric-semantic SLAM.

The introduction of discrete states into the SLAM problem greatly increases its practical difficulty. When formulated as an optimization problem, metric-semantic SLAM in its most general form is nonlinear and nonconvex, just as in purely metric SLAM, but also combinatorial, involving search over a state space whose size can be exponentially large in the number of discrete states to be estimated.

Given a model of object detections and classifications as the output of a (noisy)

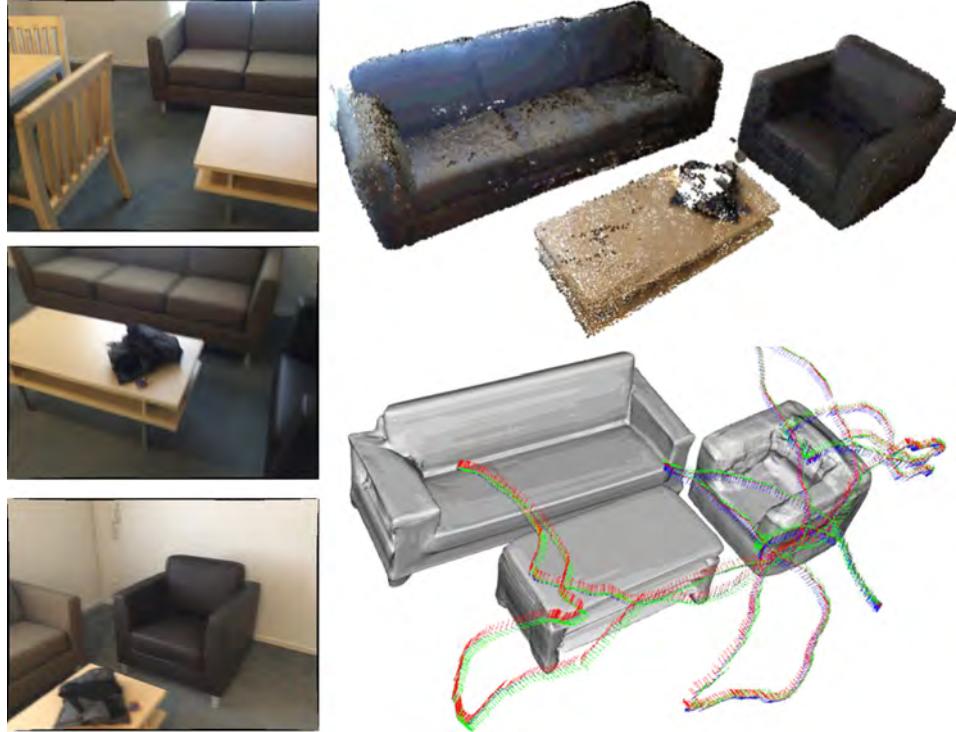


Figure 16.9 Qualitative results of 3D object pose and implicit SDF shape reconstruction on the ScanNet dataset [238] (scene 0087) using ELLIPSDF [993]. The figure shows RGB images from the scene (left), a ground-truth colored point cloud reconstruction (top right), and reconstructed object meshes using SDF models decoded from the object shape codes and optimized  $\text{SIM}(3)$  poses (bottom right).

sensor, the problem of jointly estimating the latent semantic class and geometry of landmarks in the environment can be posed in terms of MAP inference (*cf.* Chapter 1):

$$\hat{X}, \hat{L}, \hat{D} = \arg \max_{X, L, D} p(X, L, D \mid Z), \quad (16.16)$$

where  $Z$  denotes the full set of measurements (including semantic measurements);  $X$  the set of robot poses;  $L$  the set of environmental landmarks, which typically consist of some geometric information (*e.g.*, position, orientation, size, and shape) coupled with a discrete semantic label from a known, fixed set of classes; and  $D$  the set of associations between measurements in  $Z$  and landmarks in  $L$ . A key observation is that discrete-valued categorical information about objects can be naturally combined with the already discrete inference problem of data association: the knowledge of an object’s category can help distinguish it in clutter from

other objects. This formulation unifies discrete models of semantic category, geometric estimation, and data association; however, in addition to being nonconvex and high-dimensional (as in the standard SLAM formulation), it now also involves combinatorial optimization. Moreover, in committing to the use of semantics for data association, one must cope with the errors of learned perception models.

In general, the MAP inference problem in (16.18) is computationally intractable [591, Section 13.1.1]. Indeed, even the purely continuous estimation problems arising in robot perception are typically NP-hard (see Chapter 6 for a broader discussion). Despite this, smooth (local) optimization methods often perform quite well on such problems, both in their computational efficiency (owing to the fact that gradient computations are typically inexpensive) and quality of solutions when a good initialization can be supplied. However, even if we assume the ability to efficiently solve continuous estimation problems, the introduction of discrete variables complicates matters considerably: in the worst-case, solving for the joint MAP estimate globally requires that for each assignment to the discrete states we solve a continuous optimization subproblem, and discrete state spaces grow *exponentially* in the number of discrete variables under consideration. Consequently, efficient approximate solutions are needed.

It will be useful to consider the representation of eq. (16.16) in terms of its *hybrid discrete-continuous factor graph*:

$$\begin{aligned} p(\Theta, D \mid Z) &\propto \prod_k \phi_k(\mathcal{V}_k), \\ \mathcal{V}_k &\triangleq \{v_i \in \mathcal{V} \mid (\phi_k, v_i) \in \mathcal{E}\}, \end{aligned} \tag{16.17}$$

where we grouped continuous variables  $\Theta$  and discrete variables  $D$ , and each factor  $\phi_k$  is in correspondence with either a measurement likelihood of the form  $p(z_k \mid \mathcal{V}_k)$  or a prior  $p(\mathcal{V}_k)$ . In (16.17),  $\mathcal{V}_k$  is the set of (continuous or discrete) variables involved in factor  $\phi_k$ . In particular, the posterior  $p(\Theta, D \mid Z)$  can be decomposed into factors  $\phi_k$  of three possible types: *discrete* factors  $\phi_k(D_k)$  where  $D_k \subseteq D$ , *continuous* factors  $\phi_k(\Theta_k)$ ,  $\Theta_k \subseteq \Theta$ , and *discrete-continuous* factors  $\phi_k(\Theta_k, D_k)$ . In turn, the maximum *a posteriori* inference problem in (16.16) can be posed as follows:

$$\begin{aligned} \Theta^*, D^* &= \arg \max_{\Theta, D} p(\Theta, D \mid Z) \\ &= \arg \max_{\Theta, D} \prod_k \phi_k(\mathcal{V}_k) \\ &= \arg \min_{\Theta, D} \underbrace{\sum_k -\log \phi_k(\mathcal{V}_k)}_{\triangleq \mathcal{L}(\Theta, D)}. \end{aligned} \tag{16.18}$$

That is to say, we can maximize the posterior probability  $p(\Theta, D \mid Z)$  by minimizing the negative log posterior, which in turn decomposes as a summation.

It is common to consider hybrid estimation problems that can be represented in terms of nonlinear least-squares problems, which will allow the use of the tools developed in Chapter 1, like iSAM2 [533], for the estimation of continuous states of interest. In particular, we can consider discrete-continuous factors  $\phi_k(\Theta_k, D_k)$  admitting a description as:

$$\begin{aligned} -\log \phi_k(\Theta_k, D_k) &= \|r_k(\Theta_k, D_k)\|_2^2, \\ \Theta_k &\subseteq \Theta, \quad D_k \subseteq D, \end{aligned} \tag{16.19}$$

where  $r_k$  is typically nonlinear in  $\Theta$  and first-order differentiable with respect to  $\Theta$ , and the equality may be up to a constant independent of  $\Theta$  and  $D$ . Likewise, we consider factors involving only continuous variables admitting an analogous representation.

With this formulation, we can leverage the conditional independence structure of the factor graph model to develop an efficient local inference method. First, note that if we fix any assignment to the discrete states, the only variables remaining are continuous and approximate inference can be performed efficiently using smooth optimization techniques as in Chapter 1. In this sense, if we happened to know the assignment to the discrete variables, continuous optimization becomes “easy.” On the other hand, if we fix an estimate for the continuous variables, we are left with an optimization problem defined over a discrete factor graph which can be solved to global optimality using max-product variable elimination. The latter still requires exploration of *exponentially many* discrete states in the worst case, but also opens the door to well-established heuristics to compute approximate estimates.

In particular, consider a partition of the discrete states into mutually exclusive subsets  $D_j \subseteq D$  which are *conditionally independent* given the continuous states:

$$p(D | \Theta, Z) \propto \prod_j p(D_j | \Theta, Z). \tag{16.20}$$

It is straightforward to verify from the mutual exclusivity of each set  $D_j$  that the problem of optimizing the conditional in (16.20) then breaks up into subproblems involving each  $D_j$ :

$$\max_D p(D | \Theta, Z) \propto \prod_j \left[ \max_{D_j} p(D_j | \Theta, Z) \right]. \tag{16.21}$$

Critically, we have exchanged computation of the maximum of the product with the product of each maximum computed *independently*. In cases where the discrete states decompose into particularly small subsets ( $|D_j| \ll |D|$ ), inference may be carried out efficiently.

Many hybrid optimization problems in robotics admit factorizations of the form in (16.20). For example, point-cloud registration (with discrete data association variables), robust pose-graph optimization (with switch variables for outlier rejection), and metric-semantic SLAM (with data association variables and discrete

object classes) can all be formulated in a way that admits this relatively “friendly” decomposition (see, *e.g.*, [276, 616] and Section 3.3.2 in Chapter 3).

We can make use of this idea in a few ways. For example, DC-SAM [276] makes use of alternating minimization by first, fixing an initial iterate  $\Theta^{(i)}$ . Then, DC-SAM attempts to solve the following subproblems:

$$D^{(i+1)} = \arg \min_D \mathcal{L}(\Theta^{(i)}, D) \quad (16.22a)$$

$$\Theta^{(i+1)} = \arg \min_{\Theta} \mathcal{L}(\Theta, D^{(i+1)}). \quad (16.22b)$$

We may then repeat (16.22a) and (16.22b) until the relative decrease in  $\mathcal{L}(\Theta, D)$  is sufficiently small or we have reached a maximum desired number of iterations. In practice, each subproblem need not be solved to optimality. Rather, the usual techniques for optimization are used for the continuous subproblem in eq. (16.22b). Moreover, it is possible to generalize this two-stage block coordinate descent strategy by taking steps with respect to smaller groups of variables, trading off the per-step complexity with the number of optimization steps taken.

Bowman et al. [108] adopted a similar strategy for sparse metric-semantic SLAM. They performed joint optimization via expectation maximization (EM): First, we fix the robot poses and landmark locations to compute data association *probabilities* and landmark classes (the *E-step*). Next, we fix the data association probabilities and landmark classes and optimize the robot poses and landmark locations, with measurements weighted by the respective probabilities of their landmark correspondence (the *M-step*). This approach assigns “soft” associations to objects, gradually converging to a locally optimal solution of (16.16). It has also been shown that the probabilities in the E-step can be exactly recovered using a matrix permanent computation. By approximating the matrix permanent, we can in turn approximate the data association probabilities needed in the M-step more efficiently [40].

*Multi-hypothesis* methods aim to solve the optimization in eq. (16.18) by explicitly searching over possible assignments to the discrete states in  $D$ . Since the size of the search space is exponentially large in the number of discrete states, these methods use heuristics to prune the search space in order to remain computationally tractable. General approaches like MH-iSAM2 [479] and iMHS [518] have been applied to hybrid estimation problems like robust pose-graph optimization and contact estimation for legged robots, but have not yet been applied to sparse metric-semantic SLAM. Bernreiter et al. [74] present a method tailored to metric-semantic mapping and localization that makes use of a multi-hypothesis representation.

An alternative approach to the combinatorial inference problem of eq. (16.16) is to reframe the optimization over discrete variables as one over only continuous-valued variables. In early work to this end, Sünderhauf et al. [1058] optimized probabilities of semantic labels, which are defined over the  $(K - 1)$ -dimensional unit simplex for a  $K$ -class semantic labeling problem. This approach is similar to the strategies used in

Chapter 3 for outlier rejection, but with the extension to multiple labels. Similarly, there are methods that attempt to approximate the full posterior in (16.16) with all of the discrete states marginalized out, resulting in a mixture. Since this posterior distribution is generally non-Gaussian, approximation methods are used in practice (see, *e.g.*, [337, 491] for non-Gaussian solvers targeted toward SLAM applications, and [274, 41] for applications to metric-semantic SLAM).

### 16.3 Dense Metric-Semantic Representations

In this section, we discuss dense metric-semantic representations, which extend the representations reviewed in Chapter 5 to include semantic information. We start with point and surfel map representations that utilize semantic information in Section 16.3.1. Next, we discuss how voxel map representations, such as the widely used OctoMap [478], can be extended to capture semantic information in a probabilistically consistent way in Section 16.3.2. Finally, in Section 16.3.3, we discuss mesh representations which can be less expensive to store and easier to deform in response to loop closures in comparison to voxel maps.

#### 16.3.1 Point-based and Surfel-based Metric-Semantic SLAM

In this section, we focus on *metric-semantic point cloud and surfel map representations*. The availability of large-scale densely annotated datasets [68, 137, 69, 330] made it possible to learn semantic segmentation [765] and panoptic segmentation [764] directly on point clouds. As such models provide point-wise semantics and potentially also instance information, several LiDAR-SLAM approaches [187, 650, 521] exploited this semantic information directly to build metric-semantic map representations. As we already mentioned earlier in this chapter, capturing semantic information allows identifying dynamic objects and improving data association, either by exploiting semantics in finding correspondences for odometry [650, 521] or building a semantic scene descriptor for loop closure detection [592, 521].

A key challenge addressed by these methods is how to perform spatial integration of the potentially conflicting per-frame semantic segmentation of LiDAR point clouds into a unified map representation. To this end, one of the first LiDAR-based dense SLAM approaches that uses a semantic segmentation, called SuMa++ [187], enriched the surfel-based representation with a per-surfel semantic classes and confidence score of the surfel semantics. Using the semantic confidence together with the measurement uncertainty, Chen et al. [187] update the surfel confidence score such that conflicting semantic labels will lead to reduced surfel confidences. The confidence of a surfel is ultimately used to filter out low-confidence surfels that are either caused by measurement uncertainty or semantic uncertainty.

By integrating the semantics into the surfel-based map representation, the approach is able to account for conflicting semantic labels and equipped with this

*metric-semantic  
cloud map*  
*metric-semantic  
map*  
*point  
map*

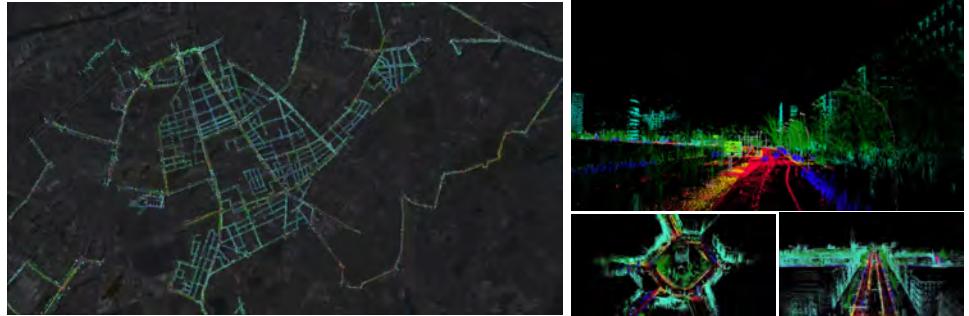


Figure 16.10 Large-scale semantic map of Berlin covering 8000 km of roads, generated by a fleet of vehicles using the method in [193] (left). Zoomed-in sections of the semantic map show fine-grained 3D reconstruction details (right). Image from [193] (©2022 IEEE).

capability, the semantic map representation can more effectively filter dynamic objects. Here, Chen et al. [187] showed that simply removing all potentially moving objects just based on the semantics is inferior to an approach that accounts only for conflicting semantic labels between the map and the current semantic interpretation of the scene. Chen et al. [187] attribute this to the prevalence of vehicles in urban environments, which can be also used for pose estimation. Thus, removing simply all potentially moving objects will also remove valuable information from the map representation that could be used for ICP alignment.

Follow-up approaches, like SA-LOAM [650] and SELVO [521], integrate semantics in the pose estimation by matching only semantically compatible features [1260], but also exploit semantics in the process of determining loop closures [592].

Building on LOAM [1266], Li et al. [650] use a point-based representation to store features for surface-like and corner-like structures in distinct semantic maps. For loop closures, the approach of this work uses a semantic graph generated from a point-wise semantic segmentation to match scenes using the semantics.

Similarly, the approach of Jiang et al. [521] also accounts for correspondences of matching semantic classes, but extend the well-known ScanContext descriptor to account for the semantic in the matching process of scene descriptors.

An example of a large-scale semantically annotated point-cloud map obtained using stereo camera images only is shown in Figure 16.10. The approach [193] uses dense 2D semantic labels from the stereo camera images and combines a direct sparse visual odometry front-end with a global optimization back-end.

### 16.3.2 Voxel-based Metric-Semantic SLAM

Dense environment models utilizing voxels have been widely used as alternatives to point cloud mapping [97, 1061, 1121, 347]. Let  $\mathcal{E}$  denote the environment in which a robot is navigating. As we have seen in Chapter 5, a voxel map  $\mathbf{m}$  is a partitioning of

$\mathcal{E}$  using a grid of non-overlapping cubes (*i.e.*, voxels), where each voxel stores a set of environment attributes, such as occupancy, temperature, etc., that are assumed to be constant within the corresponding physical 3D space. This section discusses how to augment voxel-based representations with semantic information, in addition to occupancy or distance information.

The first step towards extending voxel-based representations to metric-semantic mapping is the inclusion of semantics into the robot observations. Let the environment  $\mathcal{E}$  be divided into a collection of disjoint sets  $\mathcal{E}_k \subset \mathbb{R}^3$ , each associated with a semantic category  $k \in \mathcal{K} := \{0, \dots, K\}$ , with  $\mathcal{E}_0$  denoting the free space and each  $\mathcal{E}_k$  for  $k > 0$  representing a different object class such as building, cars, and terrain. Consider a robot equipped with a sensor that provides information about the distance to and semantic categories of surrounding objects along a set of rays  $\{\eta_b\}$ , where  $b$  is the ray index,  $\eta_b \subset \mathbb{R}^3$  with  $\|\eta_b\| = r_{\max}$ , and  $r_{\max}$  is the maximum sensing range. Note that  $\{\eta_b\}$  is just a collection of vectors of length  $r_{\max}$  that represent the rays of a ranging sensor, such as an RGB-D camera or a LiDAR. We can associate a semantically annotated point measurement with each ray by processing camera [763] or LiDAR [765] measurements with a semantic segmentation algorithm. More concretely, for a robot with orientation  $\mathbf{R}_t \in \text{SO}(3)$  and position  $\mathbf{p}_t \in \mathbb{R}^3$  at time  $t$ , a *range-category observation* is defined as a collection  $\mathcal{Z}_t = \{\mathbf{z}_{t,b}\}_b$  of range and category measurements  $\mathbf{z} := (r_{t,b}, y_{t,b}) \in \mathbb{R}_{\geq 0} \times \mathcal{K}$  along the sensor rays  $\mathbf{R}_t \eta_b + \mathbf{p}_t$ . Figure 16.11 shows an example where each pixel in an RGB image corresponds to a ray  $\eta_b$ , while its corresponding values in the semantic segmentation and range images are the category  $y_{t,b}$  and range  $r_{t,b}$ , respectively. The goal of voxel-based metric-semantic mapping is to incrementally construct a multi-class map  $\mathbf{m}$  of the environment  $\mathcal{E}$  based on streaming range-category observations. The map  $\mathbf{m}$  is modeled as a grid of cells  $i \in \mathcal{I} := \{1, \dots, N\}$ , each labeled with a category  $m_i \in \mathcal{K}$ .

Consider the PDF  $p(\mathcal{Z}_t | \mathbf{m}, \mathbf{R}_t, \mathbf{p}_t)$  that models noise in sensor observations. This observation model allows integrating the measurements into a probabilistic map representation using Bayesian updates. Let  $p_t(\mathbf{m}) := p(\mathbf{m} | \mathcal{Z}_{1:t}, \mathbf{R}_{1:t}, \mathbf{p}_{1:t})$  be the probability mass function (PMF) of the map  $\mathbf{m}$  given the robot trajectory  $(\mathbf{R}_{1:t}, \mathbf{p}_{1:t})$  and observations  $\mathcal{Z}_{1:t}$  up to time  $t$ . Given a new observation  $\mathcal{Z}_{t+1}$  obtained from robot pose  $(\mathbf{R}_{t+1}, \mathbf{p}_{t+1})$ , the Bayesian update of the map is:

$$p_{t+1}(\mathbf{m}) \propto p(\mathcal{Z}_{t+1} | \mathbf{m}, \mathbf{R}_{t+1}, \mathbf{p}_{t+1}) p_t(\mathbf{m}). \quad (16.23)$$

For brevity, we omit the dependence of the map distribution and the observation model on the robot pose throughout the rest of this analysis.

The work by Asgharivaskasi and Atanasov [37] presents an online voxel-based semantic mapping technique by generalizing the log-odds occupancy mapping algorithm [1085, Ch. 9] to multi-class maps. In particular, an explicit derivation for the Bayesian update in (16.23) is obtained using a multinomial logit model to represent the map PMF  $p_t(\mathbf{m})$  where each cell  $m_i$  of the map stores the probability of object

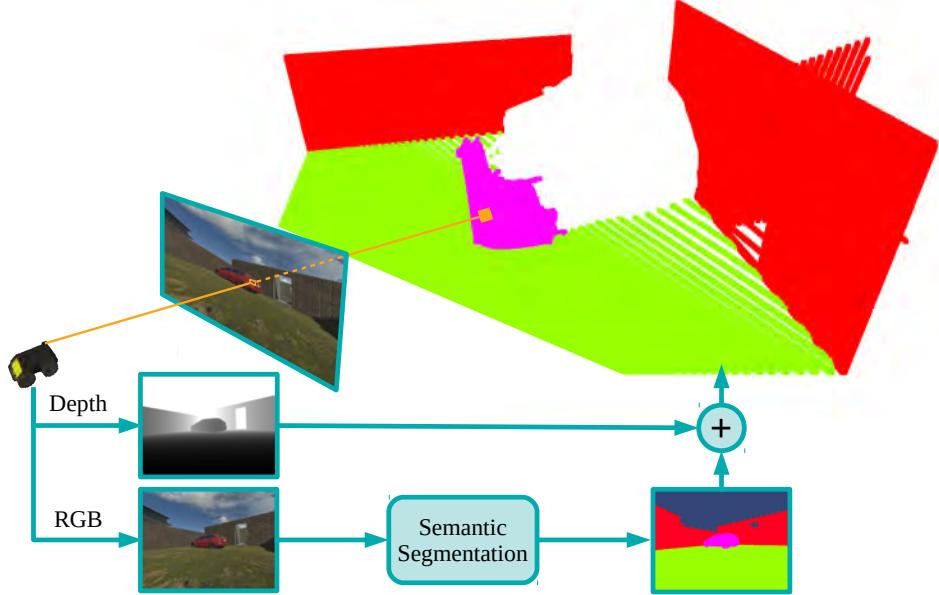


Figure 16.11 Creation process of a range-category observation. The robot is equipped with an RGB-D sensor, where the RGB part of the sensor measurement is fed to a semantic segmentation algorithm. Given the camera intrinsics, the semantic segmentation and depth images are combined to form a semantically-annotated 3D point cloud.

classes in  $\mathcal{K}$ . To ensure linear model complexity with respect to the map size  $N$ , a factorized PMF is maintained over the map cells:

$$p_t(\mathbf{m}) = \prod_{i=1}^N p_t(m_i).$$

*multi-class log odds*

The *multi-class log odds* of the PMF  $p_t(m_i)$  of an individual cell is a vector over the semantic categories  $\mathcal{K}$  that captures the log ratio of their likelihood:

$$\mathbf{h}_{t,i} := \begin{bmatrix} \log \frac{p_t(m_i=0)}{p_t(m_i=0)} & \dots & \log \frac{p_t(m_i=K)}{p_t(m_i=0)} \end{bmatrix}^\top \in \mathbb{R}^{K+1},$$

where the free-class likelihood  $p_t(m_i = 0)$  is used as a pivot. Given the log-odds vector  $\mathbf{h}_{t,i}$ , PMF of cell  $m_i$  may be recovered using the softmax function  $\sigma : \mathbb{R}^{K+1} \mapsto \mathbb{R}^{K+1}$ :

$$p_t(m_i = k) = \sigma_{k+1}(\mathbf{h}_{t,i}) := \frac{\mathbf{e}_k^\top \exp(\mathbf{h}_{t,i})}{\mathbf{1}^\top \exp(\mathbf{h}_{t,i})},$$

where  $\mathbf{e}_k$  is the standard basis vector with  $k^{\text{th}}$  element equal to 1 and 0 elsewhere,  $\mathbf{1}$  is the vector with all elements equal to 1, and  $\exp(\cdot)$  is applied element-wise to the vector  $\mathbf{h}_{t,i}$ . The Bayesian update in (16.23) for  $\mathbf{h}_{t,i}$  can then be obtained in

terms of the range-category observation model, evaluated at a new measurement set  $\mathcal{Z}_{t+1}$ :

$$\mathbf{h}_{t+1,i} = \mathbf{h}_{t,i} + \sum_{\mathbf{z} \in \mathcal{Z}_{t+1}} \mathbf{l}_i(\mathbf{z}), \quad (16.24)$$

where  $\mathbf{l}_i(\mathbf{z})$  is the observation model log-odds:

$$\mathbf{l}_i(\mathbf{z}) := \left[ \log \frac{p(\mathbf{z}|m_i=0)}{p(\mathbf{z}|m_i=K)} \quad \dots \quad \log \frac{p(\mathbf{z}|m_i=K)}{p(\mathbf{z}|m_i=0)} \right]^\top. \quad (16.25)$$

To complete the Bayesian multi-class mapping equation in (16.24) we need a particular observation model. When a sensor measurement is generated, the sensor ray continues to travel until it hits an obstacle of category  $\mathcal{K} \setminus \{0\}$  or reaches the maximum sensing range  $r_{\max}$ . The labeled range measurement  $\mathbf{z} = (r, y)$  obtained from position  $\mathbf{p}$  with orientation  $\mathbf{R}$  indicates that map cell  $m_i$  is occupied if the measurement end point  $\mathbf{p} + \frac{r}{r_{\max}} \mathbf{R} \boldsymbol{\eta}$  lies in the cell. If  $m_i$  lies along the sensor ray but does not contain the end point, it is observed as free. Finally, if  $m_i$  is not intersected by the sensor ray, no information is provided about its occupancy. A consistent observation model should hence reflect such properties in its definition, either through manual tuning [37] or learning from data [1162]. For example, the work in [37] parameterizes the observation model log-odds vector in (16.25) as a piecewise constant function along the sensor ray:

$$\mathbf{l}_i((r, y)) := \begin{cases} \phi^+ + \mathbf{E}_{y+1} \psi^+, & r \text{ indicates } m_i \text{ is occupied}, \\ \phi^-, & r \text{ indicates } m_i \text{ is free}, \\ 0, & \text{otherwise,} \end{cases} \quad (16.26)$$

where  $\mathbf{E}_k := \mathbf{e}_k \mathbf{e}_k^\top$  and  $\psi^+, \phi^-, \phi^+ \in \mathbb{R}^{K+1}$  are parameter vectors, whose first elements are 0 to ensure that  $\mathbf{l}_i(\mathbf{z})$  is a valid semantic log-odds vector. Figure 16.12 illustrates the proposed Bayesian multi-class mapping method using the observation model of (16.26).

It is important to note that utilizing a regular-grid discretization to represent  $\mathcal{E}$  has prohibitive storage and computation requirements. Large continuous portions of many real environments are unoccupied, suggesting that adaptive discretization is significantly more efficient. A popular method to improve the memory efficiency of voxel maps is proposed by Hornung et al. [478], called *OctoMap*, in which the octree data structure is utilized to combine voxels with equal occupancy probability. An octree is a hierarchical data structure containing nodes that represent a section of the physical environment. Each node has either 0 or 8 children, where the latter corresponds to the 8 octants of the Euclidean 3D coordinate system. Thus, the children of a parent node form an eight-way octant partition of the space associated with the parent node. An octree node that does not have any children is called a *leaf* node, which provides the highest-resolution visualization of an octree map. In [37], the authors extend the probabilistic 3D mapping technique of OctoMap to

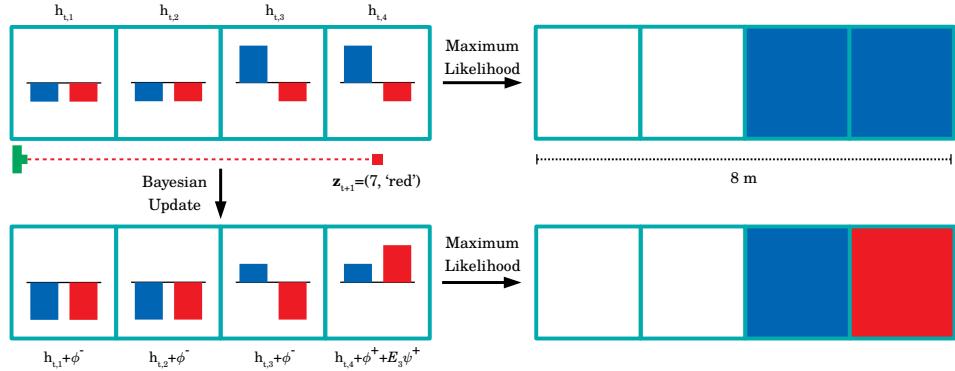


Figure 16.12 Illustration of the Bayesian multi-class mapping using a range-category sensor measurement, where the object classes are defined as  $\mathcal{K} = \{\text{'free'}, \text{'blue'}, \text{'red'}\}$ . The top-left shows the semantic log-odds vector at time  $t$  for each map cell along an observation ray, shown as the dotted red line. The top-left represents a maximum likelihood estimation of the map at time  $t$ , based on semantic log-odds vector. The range-category observation  $z_{t+1}$  contain a range measurement of 7 meters and classification of ‘red’ at the point of incidence. The bottom row shows the updated semantic log-odds vectors alongside the maximum likelihood category estimations for each cell at time  $t + 1$ .

#### *multi-class octree*

metric-semantic representations through introduction of *multi-class octree*. A multi-class octree is a type of octree data structure where each node stores a categorical probability distribution over the set of object classes  $\mathcal{K}$  in the form of a semantic log-odds vector  $\mathbf{h}_{i,t}$ . Figure 16.13 shows an example of a multi-class octree data structure.

In order to register a new observation, a ray-casting procedure over an octree (e.g. [427] or [16]) is performed to find the observed leaf nodes. Then, for each observed leaf node, if the observation demands an update, the leaf node is recursively expanded to the smallest resolution and the semantic log-odds of the downstream nodes are updated using (16.24). To obtain a compressed octree, it is necessary to define a rule for information fusion from child nodes towards parent nodes. Depending on the application, different information fusion strategies may be implemented. For example, a conservative strategy would assign the semantic log-odds of the child node with the highest occupancy probability to the parent node. Alternatively, one can simply assign the average semantic log-odds vector of the child nodes to their parent node, which is equivalent to the Bayesian fusion of information in the original probability space. The benefit of an octree representation is the ability to combine similar cells (leaf nodes) into a large cell (inner node). This is called *pruning* the octree. Every time after an observation is integrated to the map, the tree nodes are checked in a bottom-up manner to identify pruning opportunities. If the children of an inner node are all leaf nodes and have equal semantic log-odds, the children

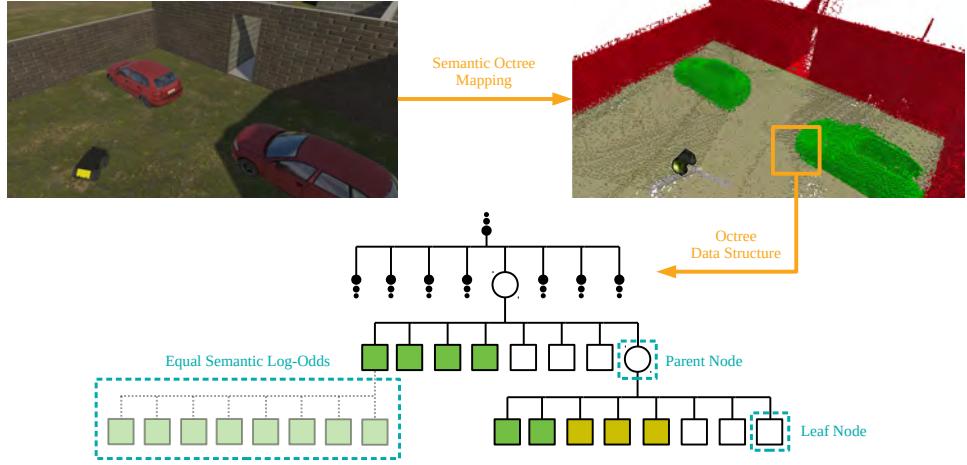


Figure 16.13 An instance of multi-class octree mapping. The top-left shows the environment in which a robot with range-category sensor performs mapping. The top-right shows the resulting multi-class octree map, where each class is shown with a distinct color (free class is transparent). The bottom depicts the octree data structure corresponding to a section of the map. The circle and square nodes represent *parent* and *leaf* nodes, respectively. Each leaf node either belongs to the smallest resolution of the octree, or contains children with identical semantic log-odds vectors.

are pruned and the inner node is converted into a leaf node with the same semantic log-odds as its children. This helps to compress the majority of the free cells into a few large cells, while the occupied cells usually do not undergo pruning since only their surfaces are observed by the sensor and their inside remains an unexplored region. Figure 16.14 displays the update and pruning procedure for a multi-class octree.

Implementations of multi-class octree mapping, such as the one in [37], consider additional design choices in order to enable real-time performance. For example, due to sensor noise, it is unlikely that cells belonging to the same class (*e.g.*, free or occupied by the same obstacle) attain identical semantic log-odds. Maximum and minimum limits for the elements of the semantic log-odds are used so that each cell arrives at a stable state as its semantic log-odds entries reach the limits. Stable cells are more likely to share the same multi-class probability distribution, consequently increasing the chance of octree pruning. However, thresholding causes loss of information near  $p_t(m_i = k) = 1$ ,  $k \in \mathcal{K}$  which can be controlled by the maximum and minimum limits. Furthermore, the space and computational complexity of multi-class map updates increases linearly with respect to the number of class labels  $K$ , while the probability of pruning reduces exponentially as  $K$  grows. Hence, one may store the semantic log-odds for only the  $\bar{K}$  most likely class labels, and lump the rest of the labels into a single *others* variable. This encourages more

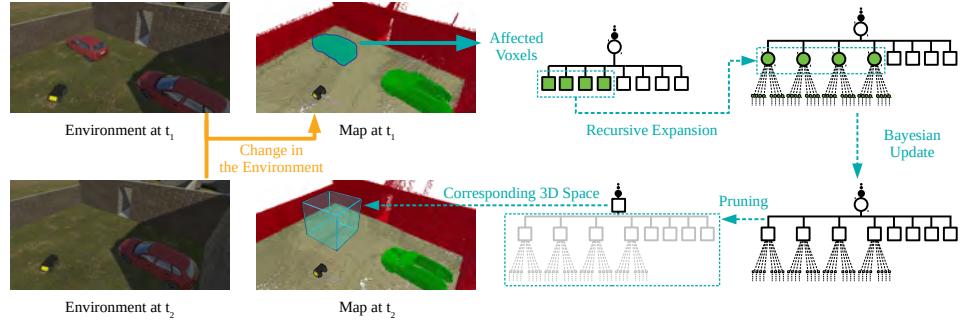


Figure 16.14 Multi-class octree map update. A change in the environment between  $t = t_1$  and  $t = t_2$  triggers a map update. Through ray-casting, the voxels highlighted in cyan need to be updated. For this aim, the corresponding nodes in the octree are recursively expanded, and updated via (16.24). Simultaneously, pruning opportunities are found for the children of each parent node. The pruning process leads to smaller, but physically larger, octree nodes, shown as the cyan cube for the map at  $t = t_2$ .

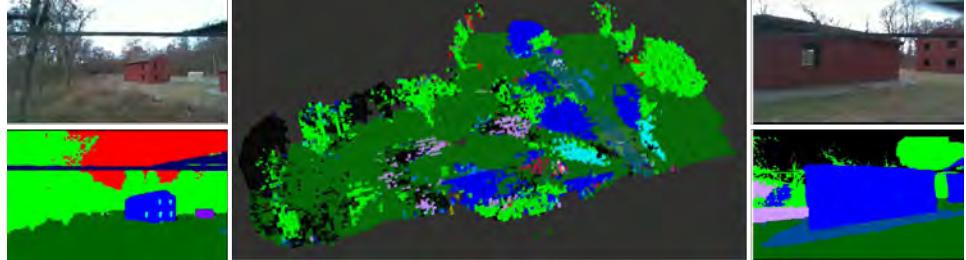


Figure 16.15 Qualitative results from a real-world SLAM experiment in which two quadrotor robots construct a multi-class octree map using the ROAM algorithm [38]. RGB and semantically segmented images from RGB-D cameras onboard the two robots are shown on the left and right. The middle plot shows the resulting multi-class octree map.

frequent pruning and significantly reduces the computation complexity during map updates in cases with many semantic categories.

Several robots can construct a multi-class octree map collaboratively. Riemannian Optimization for Active Mapping (ROAM) [38] formulates an optimization problem over a graph with node variables representing octree maps of different robots and a consensus constraint requiring the robots to achieve agreement on their local maps. ROAM is based on distributed Riemannian optimization relying only on one-hop communication to achieve consensus and optimality guarantees. An example of multi-class octree mapping with the ROAM algorithm [993] applied to real-world data is shown in Figure 16.15.

### 16.3.3 Mesh-based Metric-Semantic SLAM

While voxel-based maps provide an effective representation to probabilistically fuse information over time, they have two downsides. First, they tend to be expensive to store (an issue that can be partially mitigated using octree data structures). Second, they cannot be easily edited in response to loop closures: loop closures entail large deformations in the robot trajectory estimate and the corresponding map, and voxel-based maps are expensive to update in response to these deformations. Updating dense volumetric representations typically involves de-integrating and re-integrating previous observations, which is computationally expensive (*e.g.*, [237]).

In this section we discuss mesh-based representations as a way to circumvent both issues. Meshes are collections of polygons (typically triangles) and their storage complexity scales with the complexity of the scene being represented. Rather than being directly built from sensor data, it is fairly common to build a voxel-based representation first, and then compute a mesh from it using standard algorithms (*e.g.*, marching cubes). Advanced approaches, such as [940], build on ideas used in geometric SLAM [1180], and only build a voxel-based map in a spatial window around the robot, while the voxel-based representation is incrementally converted into a mesh as voxels leave this active window. More interestingly, the mesh representation enables to efficiently deform the map in response to loop closures, hence providing a more computationally effective way to correct the map compared to voxel de-integration and re-integration. We now review how to deform mesh-based representations, an approach called Pose-Graph and Mesh Optimization (PGMO) [940].

PGMO builds on the notion of *embedded deformation graph*, which was first introduced in the context of shape manipulation in computer graphics [1046]. The deformation graph is a collection of *control nodes*, obtained by downsampling the original mesh. Each control node is attached a local coordinate frame and the overall graph can be deformed by solving an optimization problem that enforces local rigidity of the graph edges. Then, after solving this optimization that computes the deformation of the control nodes, the rest of the mesh is interpolated back, using the newly found control nodes' configurations. We formalize the approach below.

For dense 3D metric-semantic SLAM, a deformation graph can be created from the pose-graph of robot poses and a downsampled version of the dense mesh. The pose vertices of the deformation graph are defined as the nodes of the robot pose graph with associated transformation  $\mathbf{X}_i = [\mathbf{R}_i^X, \mathbf{t}_i^X]$ , and are connected according to the connectivity of the pose graph. The mesh vertices of the deformation graph are defined as the vertices of the down-sampled mesh with associated transformation  $\mathbf{M}_k = [\mathbf{R}_k^M, \mathbf{t}_k^M]$  for mesh vertex  $k$ , and are connected according to the edges of the simplified mesh. A pose vertex  $i$  is connected to the mesh vertex  $k$  if the mesh vertex  $k$  is visible to the sensor associated to pose  $i$ . Figure 16.16 shows the components of the deformation graph.

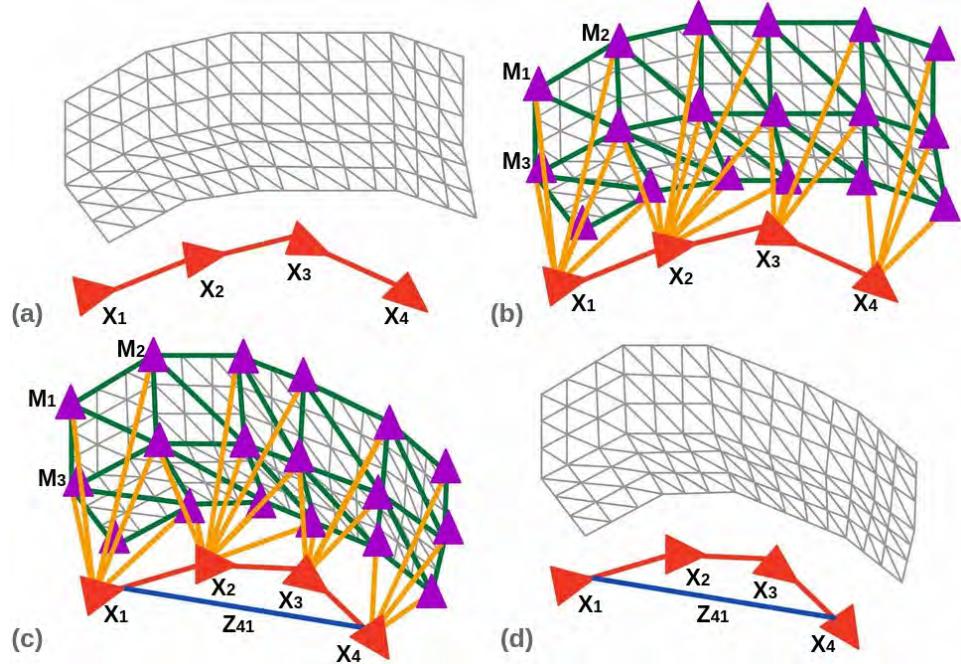


Figure 16.16 PGMO deformation. (a) Un-deformed pose graph and mesh. (b) Creating a deformation graph. Pose vertices are shown in red and mesh vertices are shown in purple. The green edges are the edges describing the connectivity of the simplified mesh. The yellow edges are the edges connecting the pose vertices to the mesh vertices based on visibility from the sensor. (c) A loop closure triggers the pose graph and mesh optimization.  $\mathbf{X}_i$  and  $\mathbf{M}_i$  are updated based on the optimization results and the dense mesh deformed. (d) The resulting deformed mesh and pose graph.

In the un-deformed state,  $\mathbf{X}_i$  is the un-optimized pose for node  $i$ , while  $\mathbf{R}_k^M = \mathbf{I}_3$  and  $\mathbf{t}_k^M = \mathbf{g}_k$ , where  $\mathbf{g}_k$  is the original world frame position of vertex  $k$ . Intuitively, these transformations describe the local deformations on the mesh:  $\mathbf{R}_k^M$  is the local rotation centered at vertex  $k$  while  $\mathbf{t}_k^M - \mathbf{g}_k$  is the local translation. Once a deformation is triggered (*e.g.*, loop closure detected), the deformation graph is optimized

as follows,

$$\begin{aligned} \arg \min_{\substack{\mathbf{X}_1, \dots, \mathbf{X}_n \in \text{SE}(3) \\ \mathbf{M}_1, \dots, \mathbf{M}_m \in \text{SE}(3)}} & \sum_{\mathbf{Z}_{ij}} \|\mathbf{X}_i \mathbf{Z}_{ij} - \mathbf{X}_j\|_{\Omega_{ij}}^2 \\ & + \sum_{k=0}^m \sum_{l \in \mathcal{N}^M(k)} \|\mathbf{R}_k^M(\mathbf{g}_l - \mathbf{g}_k) + \mathbf{t}_k^M - \mathbf{t}_l^M\|_{\Omega_{kl}}^2 \\ & + \sum_{i=0}^n \sum_{l \in \mathcal{N}^M(i)} \|\mathbf{R}_i^X \tilde{\mathbf{g}}_{il} + \mathbf{t}_i^X - \mathbf{t}_l^M\|_{\Omega_{il}}^2, \end{aligned} \quad (16.27)$$

where  $\mathcal{N}^M(i)$  indicates the neighboring mesh vertices to a vertex  $i$  in the deformation graph,  $\mathbf{g}_i$  denotes the non-deformed (initial) position of mesh vertex  $i$  in the deformation graph, and  $\tilde{\mathbf{g}}_{il}$  denotes the non-deformed position of vertex  $l$  in the local coordinate frame of the odometric pose of node  $i$ , such that  $\tilde{\mathbf{g}}_{il} = \tilde{\mathbf{R}}_i^\top (\mathbf{g}_l - \mathbf{g}_i)$ , where  $\tilde{\mathbf{R}}_i$  is the initial rotation of pose vertex  $i$  in the deformation graph.

Notice that the first term of (16.27) follows from generic PGO, the second term enforces local rigidity between mesh vertices (by penalizing deformations for mesh vertices connected by an edge), and the third term enforces the local rigidity between a pose vertex  $i$  and a mesh vertex  $l$ . Note that for a matrix  $\mathbf{A}$ ,  $\|\mathbf{A}\|_\Omega^2$  indicates the (squared) weighted Frobenius norm

$$\|\mathbf{X}\|_\Omega^2 = \text{tr}(\mathbf{X} \Omega \mathbf{X}^\top), \quad \text{with } \Omega = \begin{bmatrix} \omega_R \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \omega_t \end{bmatrix}, \quad (16.28)$$

for non-negative weights  $\omega_R$  and  $\omega_t$ , while for a vector  $\mathbf{v}$ ,  $\|\mathbf{v}\|_\Omega^2$  is the usual squared Mahalanobis norm  $\|\mathbf{v}\|_\Omega^2 = \mathbf{v}^\top \Omega \mathbf{v}$ .

Following [940], we can show (16.27) can be formulated as a PGO problem. Defining  $\mathbf{G}_{ij}$  and  $\bar{\mathbf{G}}_{ij}$  as

$$\mathbf{G}_{ij} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{g}_j - \mathbf{g}_i \\ 0 & 1 \end{bmatrix}, \quad \bar{\mathbf{G}}_{ij} = \begin{bmatrix} \mathbf{I}_3 & \tilde{\mathbf{g}}_{il} \\ 0 & 1 \end{bmatrix}, \quad (16.29)$$

we can then rewrite (16.27) as

$$\begin{aligned} \arg \min_{\substack{\mathbf{X}_1, \dots, \mathbf{X}_n \in \text{SE}(3) \\ \mathbf{M}_1, \dots, \mathbf{M}_m \in \text{SE}(3)}} & \sum_{\mathbf{Z}_{ij} \in \mathcal{Z}} \|\mathbf{X}_i \mathbf{Z}_{ij} - \mathbf{X}_j\|_{\Omega_{ij}}^2 + \\ & \sum_{\mathbf{G}_{ij} \in \mathcal{G}} \|\mathbf{M}_i \mathbf{G}_{ij} - \mathbf{M}_j\|_{\Omega_{\mathbf{G}_{ij}}}^2 + \\ & \sum_{\bar{\mathbf{G}}_{ij} \in \bar{\mathcal{G}}} \|\mathbf{X}_i \bar{\mathbf{G}}_{ij} - \mathbf{M}_j\|_{\Omega_{\bar{\mathbf{G}}_{ij}}}^2, \end{aligned} \quad (16.30)$$

where  $\mathcal{Z}$  is the set of all pose-graph edges (the red and blue edges in Figure 16.16),  $\mathcal{G}$  is the set edges from the simplified mesh (the green edges in Figure 16.16), and  $\bar{\mathcal{G}}$  is the set of all the edges connecting a pose vertex to a mesh vertex (the yellow edges

in Figure 16.16). We only optimize over translation in the second and third term, and as such  $\Omega_{G_{ij}}$  and  $\Omega_{\bar{G}_{ij}}$  correspond to information matrices where the rotation part (namely  $\omega_R$  in (16.28)) is set to zero. Taking it one step further and observing that the terms are all based on the edges in the deformation graph, we can define  $\mathbf{T}_i$  as the transformation of pose or mesh vertex  $i$  and  $\mathbf{E}_{ij}$  is the transformation that corresponds to an edge in the deformation graph that is of the form  $\mathbf{Z}_{ij}$ ,  $\mathbf{G}_{ij}$ ,  $\bar{\mathbf{G}}_{ij}$  depending on the type of edge. We are then left with the classic PGO problem,

$$\arg \min_{\mathbf{T}_1, \dots, \mathbf{T}_{n+m} \in \text{SE}(3)} \sum_{\mathbf{E}_{ij}} \|\mathbf{T}_i \mathbf{E}_{ij} - \mathbf{T}_j\|_{\Omega_{ij}}^2. \quad (16.31)$$

After the optimization, the position of each mesh vertex is updated from the affine transformations of the  $m$  nearest control nodes in the deformation graph as

$$\tilde{\mathbf{v}}_i = \sum_{j=1}^m w_j(\mathbf{v}_i) [\mathbf{R}_j^M (\mathbf{v}_i - \mathbf{g}_j) + \mathbf{t}_j^M], \quad (16.32)$$

where  $\mathbf{v}_i$  indicates the original position and  $\tilde{\mathbf{v}}_i$  the new deformed position of the  $i$ -th mesh vertex. The weights  $w_j$  assigned to each control node are defined as

$$\begin{aligned} w_j(\mathbf{v}_i) &= \frac{\bar{w}_j(\mathbf{v}_i)}{\sum_{k=1}^m \bar{w}_k(\mathbf{v}_i)} \\ \bar{w}_j(\mathbf{v}_i) &= \left(1 - \frac{\|\mathbf{v}_i - \mathbf{g}_j\|}{d_{\max}}\right)^2 \end{aligned} \quad (16.33)$$

where  $d_{\max}$  is the distance to furthest of the  $m$  control nodes from the mesh vertex.

#### 16.4 Hierarchical Metric-Semantic Representations and 3D Scene Graphs

Up to this point, we have discussed sparse representations in Section 16.2 and dense representations in Section 16.3 for encoding semantic information into the maps that our robots build. Both representations have different advantages; for instance, sparse representations present a convenient interface for manipulating objects while dense volumetric representations excel at encoding information about free-space and traversability. This section presents hierarchical representations that combine dense and sparse maps into a unified model. In particular, we first discuss how choices of spatial representations and semantic categories impact the memory required by a robot. Then we observe that organizing information hierarchically leads to memory and computational savings. Finally, we provide a more concrete example of a metric-semantic hierarchical map representations, namely *3D scene graphs*.

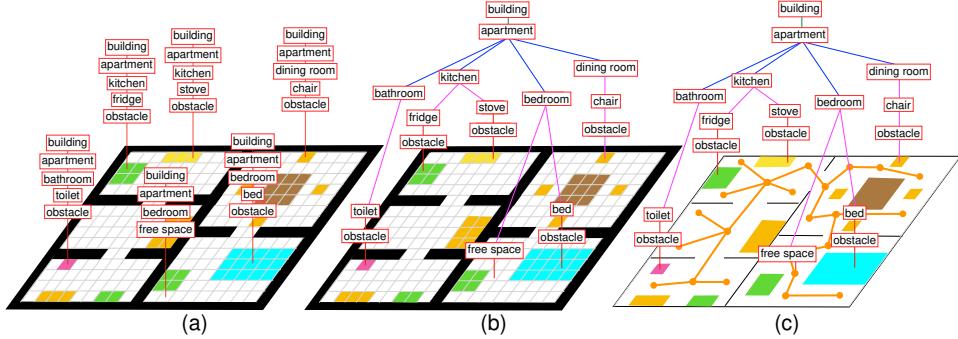


Figure 16.17 Groundings of symbols of interest to various sub-symbolic representations.

#### 16.4.1 Hierarchical Representations and Symbol Grounding

We start by taking a step back and discussing the relation between metric-semantic mapping and the *symbol grounding* problem [431]. So far, we have primarily considered semantic features that are closed-set semantic categories provided by some learned model, based on an input camera image or LiDAR scan. These features are groundings of *symbols*, since they associate concepts that have specific meanings for a human<sup>1</sup> to the robot’s sensor data. High-level instructions issued by humans, such as “go and pick up the chair”, intrinsically involve symbols. For the robot to correctly execute the instruction “go and pick up the chair”, the robot needs to ground the symbol ‘chair’ to the physical location it is situated in. This problem of embedding symbols into a map is commonly known as the *symbol grounding* problem.

In principle, we could design the perception system of our robot to ground symbols directly in posed sensor data. For instance, a robot with a camera could map image pixels to appropriate symbols (*e.g.*, ‘chair’, ‘furniture’, ‘kitchen’, ‘apartment’).<sup>2</sup> However, grounding symbols directly into sensor data is not scalable as sensor data (*e.g.*, images) is collected at high rates and is relatively expensive to store. This is neither convenient nor efficient when grounding symbols for long-term operation. Instead, we need intermediate (or *sub-symbolic*) representations that compress the raw sensor data into a more compact format, which can be used to ground symbols. In this light, the previous sections of this chapter have been concerned with choices of geometric (*i.e.*, sub-symbolic) representations and algorithms for grounding semantic information into these sub-symbolic representations.

**Hierarchy and Memory.** The most naive strategy for grounding symbols in a sub-symbolic representation is to simply store whether or not a specific symbol (out of the  $L$  symbols of interest in the scene) is present in any given voxel. This is

<sup>1</sup> *e.g.*, the word ‘chair’ is a symbol in the sense that as humans we understand what a chair is.

<sup>2</sup> Dense 2D semantic segmentation networks are examples of this strategy.

subtly different than the voxel-based semantic map discussed in Section 16.3 as we may not be willing to make the assumption that symbols are disjoint, *i.e.*, multiple symbols may be grounded to a single voxel (*e.g.*, we might say that a voxel belong to a chair, but it is also part of a kitchen). At a resolution  $\delta$ , such a voxel grid representing a scene with volume  $V$  has a memory requirement of

$$m = \mathcal{O}(L \cdot V/\delta^3). \quad (16.34)$$

This is shown in the leftmost subfigure of Figure 16.17: here we simply have a voxel-based map, where each voxel can store up to  $L$  semantic attributes.

Often the symbols that we are interested in display some sort of hierarchy. For indoor environments, we may be interested in the presence of objects and rooms (among other categories of concepts, such as floors). Instead of densely and redundantly storing these symbols in every voxel, we can rearrange the representation according to their hierarchy. Intuitively, we can organize the representation into a graph, where nodes can represent objects, rooms, or buildings, and edges represent inclusion: the children nodes of a building are the rooms contained in that building, while the children nodes of a room are the objects contained within. This is pictured in the middle subfigure of Figure 16.17. This hierarchical organization already reduces memory requirement to

$$m = \mathcal{O}(V/\delta^3 + N_{\text{objects}} + N_{\text{rooms}} + \dots + N_{\text{buildings}}), \quad (16.35)$$

where  $N_{\text{objects}}$ ,  $N_{\text{rooms}}$ ,  $N_{\text{buildings}}$  denote the number of objects, rooms, buildings visited by the robot. Importantly, this memory compression is lossless; the original dense exhaustive voxel grid representation can easily be recovered from the hierarchical representation [496].

As seen previously in this chapter, having a dense and high-resolution voxel grid of an entire scene is often impractical. If instead of using standard voxel-based representations, we use more memory-efficient sub-symbolic representations, such as an octree or an approximate clustering of free space, we can reduce the memory requirement to

$$m = \mathcal{O}(N_{\text{sub-sym}} + N_{\text{objects}} + N_{\text{rooms}} + \dots + N_{\text{buildings}}) \quad (16.36)$$

where  $N_{\text{sub-sym}}$  is often much smaller than  $V/\delta^3$ . This is often a “lossy” compression, and the original voxel-based sub-symbolic representation may not be perfectly recoverable even if the symbolic portion of the representation is [496]. This is shown in the rightmost subfigure of Figure 16.17, where we cluster the free space in a topological graph of *places* as in [496].

**Hierarchy and Inference.** In addition to being memory efficient, the hierarchical representation that we have outlined can also have important computational properties that make it more advantageous than a strictly ‘flat’ representation. To explore this, we first need to formalize the notion of hierarchical representation by introducing the definition of hierarchical graph.

**Definition 16.2** (Hierarchical Graph) A graph  $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$  is said to be hierarchical if there exists a partition of the vertices  $\mathcal{V}$  into  $l$  layers (*i.e.*,  $\mathcal{V} = \cup_{i \in 1:l} \mathcal{V}_i$ ) satisfying the following properties

- 1 **Single Parent.** Given a child node  $v \in \mathcal{V}_i$  and a parent node  $u \in \mathcal{V}_{i+1}$  that share an edge  $(v, u) \in \mathcal{E}$ , there is no other edge  $(v, u')$  where  $u' \in \mathcal{V}_{i+1}$ .
- 2 **Locality.** Every interlayer edge must be between consecutive layers, or formally no edge  $(u, v) \in \mathcal{E}$  where  $u \in \mathcal{V}_i$  and  $v \in \mathcal{V}_j$  exists such that  $|i - j| > 1$ .
- 3 **Disjoint Children.** For every node  $u, v \in \mathcal{V}_{i+1}$ , we have that the children  $\mathcal{C}(u)$  of  $u$  and  $\mathcal{C}(v)$  of  $v$  are disjoint, *i.e.*, there is no edge connecting any of the children. The set of children of a node  $s \in \mathcal{V}_{i+1}$  is defined as  $\mathcal{C}(s) \triangleq \{t : t \in \mathcal{V}_i \wedge (s, t) \in \mathcal{E}\}$ .

Note that this definition uses an assumed ordering of the layers, where 1 is the lowest layer and  $l$  is the highest. This definition implies that for any node  $v \in \mathcal{V}_i$  in the graph, the tree of descendants  $v$  is disjoint from the tree of descendants of *any* other node in  $\mathcal{V}_i$ . As shown by Hughes et al. [496], this property allows for the existence of a hierarchical tree decomposition and a corresponding bound of the treewidth of any hierarchical graph. The resulting bound is given by

$$\max_{v \in \mathcal{V}} \text{tw}[\mathcal{G}[\mathcal{C}(v)]] + 1, \quad (16.37)$$

where  $\mathcal{G}[\mathcal{C}(v)]$  is the subgraph of  $\mathcal{G}$  formed by the children of  $v$ , and  $\text{tw}[\mathcal{G}]$  is the treewidth of the graph  $\mathcal{G}$ .<sup>3</sup> The treewidth is an important property that influences the computational complexity of performing inference over the graph. The treewidth is also a well-known measure of complexity for many problems on graphs [93, 247, 320, 403]. In particular, while inference is NP-hard in general for inference on graphical models [230], proving that a graph has small treewidth opens the door to efficient, polynomial-time inference algorithms. The bound provided in (16.37) therefore allows for efficient inference over hierarchical graphs if the treewidth of descendant subgraphs remains small; this appears to hold in practice for a broad class of hierarchical graphs called 3D scene graphs [496].

We conclude this section by observing that while hierarchical representations have recently attracted a large amount of attention, hierarchical maps have been pervasive in robotics since its inception [610, 172, 1087]. Early work focused on 2D maps and investigated the use of hierarchical maps to resolve the divide between metric and topological representations [953, 355, 1263, 213, 65]. These works preceded the “deep learning revolution” and could not leverage the rich semantics currently accessible via deep neural networks.

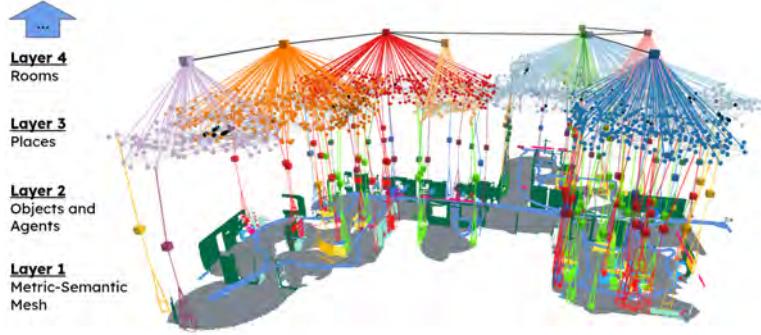


Figure 16.18 Example 3D scene graph produced by [496].

#### 16.4.2 3D Scene Graphs

The definition of a *3D scene graph* varies slightly across different research communities. The concept of 3D scene graph as a hierarchical map representation was first introduced by Armeni et al. [36] as a hierarchical representation of semantic and spatial information, including objects and rooms. Kim et al. [576] independently introduced 3D scene graphs as a graph of objects and their relationships, extending 2D scene graphs. Wald et al. [1138] constructed similar 3D scene graph of objects and their relationships from dense scene reconstructions, eventually as part of a online SLAM system [1195], adding an implicit hierarchy to [576]. Rosinol et al. [940] proposed a 3D scene graph structure that also included dynamic entities (such as humans) in addition to objects and a representation of free-space (referred to as *places*). Hughes et al. [496] constructed the first online system to produce fully hierarchical scene graphs that included objects, places, and rooms; the proposed system was able to correct the 3D scene graph to be globally consistent. An example of a 3D scene graph produced by [496] is shown in Figure 16.18.

*3D scene graph*

Formally, we define a *3D scene graph* as a graph  $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$  where each node  $v \in \mathcal{V}$  is associated with a corresponding node feature or properties  $x_v$  and each edge  $e \in \mathcal{E}$  is associated with a corresponding edge feature or properties  $x_e$ . Every 3D scene graph spatially grounds symbols, and as such, every node feature  $x_v$  is assumed to contain positional information.<sup>4</sup> Often, it makes sense to treat the scene graph  $\mathcal{G}$  as a layered graph such that  $\mathcal{V}$  is partitioned into  $l$  layers, i.e.,  $\mathcal{V} = \cup_{i \in 1:l} \mathcal{V}_i$ , where 3D scene graphs of objects and their relationships such as [576, 1138, 1195] become one layer of a more complex hierarchical representation. Approaches frequently construct 3D scene graphs from dense metric [1138, 1195]

<sup>3</sup> The bounds given in [496] have an extra term for the treewidth of the top layer  $l$ , but we assume a  $\mathcal{V}_l$  is comprised of a single node for simplicity.

<sup>4</sup> Different domains may make different modeling choices, i.e.,  $\mathbb{R}^3$  is a natural representation for 3D scene graphs of indoor environments, whereas latitude, longitude, and elevation may be a better fit for 3D scene graphs targeted at autonomous driving such as [396].

or metric-semantic [36, 940, 496] representations by clustering or segmenting the representation to obtain objects, creating a sparse, compressed grounding of object symbols that is often more computationally tractable to work with than the underlying dense representation. Additional layers, such as the layer of places or rooms in [496], are sparse representations of the free-space.

Inter-layer edges (edges between nodes in different layers) are often used in conjunction with room and place layers to encode spatial containment (*e.g.*, an edge between a room and a specific object denotes that the object is in the room), which allows for the construction of hierarchical graphs matching and efficient inference [496]. Edges may encode other spatial relationships such as *on top of* or *nearby* (often between objects) [1195]. These relationships may be directed (*e.g.*, an edge encoding a relationship of *on top of* between a book and a table has a different meaning depending on the direction of the edge). Note that certain relationship categories, such as *left/right of* or *behind*—which are common in 2D scene graphs—are view-dependent and hence less meaningful in 3D scene graphs.

**Factor Graphs and 3D Scene Graphs.** As we discussed in Section 16.3.3, it is important to efficiently correct the map representation in response to loop closures. Hughes et al. [496] propose an approach to jointly optimize the 3D scene graph and the underlying dense representation that extends the PGMO approach outlined in Section 16.3.3. In addition to the pose-graph and mesh control points, the layer of places is included in (16.27) and jointly optimized, providing a structural prior in addition to the factors between the mesh control nodes. The solution to this optimization is then used to interpolate and refine the other layers rather than reconstructing them from scratch. Other approaches such as [61] propose additional factors between the different layers of the 3D scene graph that allow them to optimize the *entire* 3D scene graph to correct for drift without requiring additional refinement steps afterwards. This however means that the optimization problem can no longer be formulated as a PGO problem.

## 16.5 Further Readings & Recent Trends

**Multi-Modal Implicit Maps, Task Planning, and Action Models.** An emerging trend is to generalize metric-semantic mapping beyond geometric, photometric, and semantic representations by encoding features from large vision-language models (VLMs). Recent approaches encode high-dimensional, language-grounded features that enable robots to reason about object affordances and robot tasks. Embedding language features, such as CLIP embeddings [896], allows locating objects using an open vocabulary and opens up potentially new avenues of building more flexible robotic systems. In particular equipping neural scene representation, such as NeRF and Gaussian Splatting, with language embeddings has recently attracted considerable interest [554]. For instance, Online Language Splatting [542] builds dense, language-aware scene representations in real time by fusing CLIP fea-

tures into Gaussian splats, enabling downstream queries and grounding tasks. We discuss many of these approaches in the next chapter.

Turning the information encoded in a semantic map into action [826, 615] is another recent development that is also enabled by encoding semantics directly into the map representation by using language-grounded embeddings. ATLAS Navigator [826] actively constructs spatial maps embedded with language features to facilitate efficient navigation to object-centric goals described in natural language. M3 [1310] introduces a persistent 3D-spatial multimodal memory that stores VLM-aligned features and supports retrieval-based reasoning for long-horizon tasks. Moving beyond mapping, LUMOS [795] integrates world modeling with language-conditioned imitation learning, allowing robots to simulate, imagine, and plan in VLM-grounded latent spaces. Finally, GNFactor [1262] demonstrates how generalizable neural feature fields can bridge multi-task robot learning with real-world generalization by encoding rich perceptual priors that inform control. These works illustrate a shift toward actionable representations, where spatial memory, language, and multimodal perception are tightly coupled to support task planning and skill execution.

However, often these maps are built as a post-processing step and an open challenge is to build such representations incrementally [542, 655], where aggregating ambiguous information from multiple varying viewpoints in a consistent and instance-aware way remains a formidable research challenge. Coupling such representations with uncertainty seems like a potential way forward, which would allow modeling ambiguities in a principled way.

**Frontiers of 3D Scene Graphs.** 3D scene graphs have been adopted for planning [17, 902, 910], manipulation [519], map compression [166], prediction [393, 690], loop closure detection [496], and localization [529], among other problems. At the same time, there is a growing body of work extending this representations in several directions. First, extending 3D scene graphs to unstructured and outdoor scenes is still an active research area, since the layers in the scene graph are harder to define and build in generic outdoor environments [73, 1034]. A related problem is that the layers of the scene graph are typically hard-coded by a human user, while ideally we would like for the robot to automatically reorganize its hierarchical representation depending on the task it is assigned; early works focusing on building more fluid task-driven 3D scene graphs include [720, 168]. A third direction consists in extending scene graphs beyond closed-set semantics, by incorporating language (or language embeddings) into 3D scene graphs [1179, 407, 720, 168]; we will discuss open-set semantics mapping approaches at length in the next chapter. A forth direction focuses on extending 3D scene graphs with additional semantics information, including motion, dynamics, and affordances [1265]. Finally, while there are established ways to perform uncertainty quantification in traditional SLAM problems (*cf.* Chapter 6), performing uncertainty quantification in hierarchical representations mixing discrete and continuous variables is still a largely unexplored problem.

**Multi-Robot Semantic Understanding.** While in this chapter we have mostly focused on single-robot metric-semantic SLAM approaches, the recent literature has also investigated extensions of these representations to multi-robot teams, where multiple robots explore an unknown environment and have to build a unified metric-semantic map representation. This setting is challenging since now the information is collected in a distributed way (hence creating communication bottlenecks when sharing data among robots). Moreover, the amount of collected data and the size of the resulting problems become unmanageable by a single robot and either require a powerful central server or a distributed solution where the workload is shared across multiple robots. Finally, in multi-robot SLAM problems one no longer has a reliable initial guess for the poses of all robots (intuitively, all robots might use a different coordinate frame), hence creating additional challenge in terms of robustness and outlier rejection (*cf.* Chapter 3 and Chapter 6). Kimera-Multi [1092] is the first approach to distributed multi-robot metric-semantic SLAM, while [1089] provides a detailed experimental analysis discussing tradeoffs between centralized and distributed SLAM approaches. Finally, recent work has extended 3D scene graph construction to multi-robot teams [167, 325].

# 17

## Towards Open-World Spatial AI

Liam Paull, Sacha Morin, Dominic Maggio, Martin Büchner, Cesar Cadena,  
Abhinav Valada, and Luca Carlone

In previous chapters, we have seen many different types of map representations, including *sparse* representations that model the world as a discrete and sparse set of landmarks (Chapter 1), *dense* representations that store detailed geometric information about 3D space (Chapter 5), as well as *hierarchical* representations, such as 3D scene graphs, that attempt to flexibly balance these two extremes (Chapter 16). Furthermore, we have seen how these representations are created in a way that is dependent on the input sensor modality, *e.g.*, vision (Chapter 7), LiDAR (Chapter 8), and radar (Chapter 9), among others. However, the types of information contained within those representations have been largely restricted to geometrical (or possibly photometric, *e.g.*, Chapter 14) properties of the environment.

In Chapter 16, we have seen that modern SLAM systems can move beyond mere geometry to encode semantics and thus enable some form of *spatial AI*, or reasoning in a more abstract and high-level fashion. This has the potential to enable robots to perform more complex tasks that can be specified at a more intuitive level. For example, goals can be specified using high-level natural language as opposed to low-level geometric instructions. This is enabled by representations that contain more semantically meaningful information (such as objects). These representations tend to be built by querying perceptual models that are trained with deep learning. However, in Chapter 16, the models used to build these representations were typically trained in a *closed-set* manner, meaning that the dictionary of potential classes, object types, or concepts, was pre-determined at the time the perceptual model (*e.g.*, object detector) was trained and is therefore fixed and finite.

In recent years, we have seen the rise of “foundation models” that are trained on massive datasets, and have shown the ability to learn *open-world* (*a.k.a.*, *open-set* or *open-vocabulary*) *representations*, meaning that they potentially generalize to any class, object type, or concept that one might encounter. These open-world representations can be useful for incorporating semantic information inside of the SLAM map, as shown in Figure 17.1.

In this chapter, we explore how foundation models can be leveraged within the context of SLAM. In particular, Section 17.1 contextualizes the chapter within the recent and growing work on foundation models and introduces basic terminology.

Section 17.2 reviews key examples of multi-modal foundation models. Section 17.3 discusses recent use of foundation models and open-set semantics for SLAM and Spatial AI. Finally, Section 17.4 concludes the chapter by discussing recent trends.

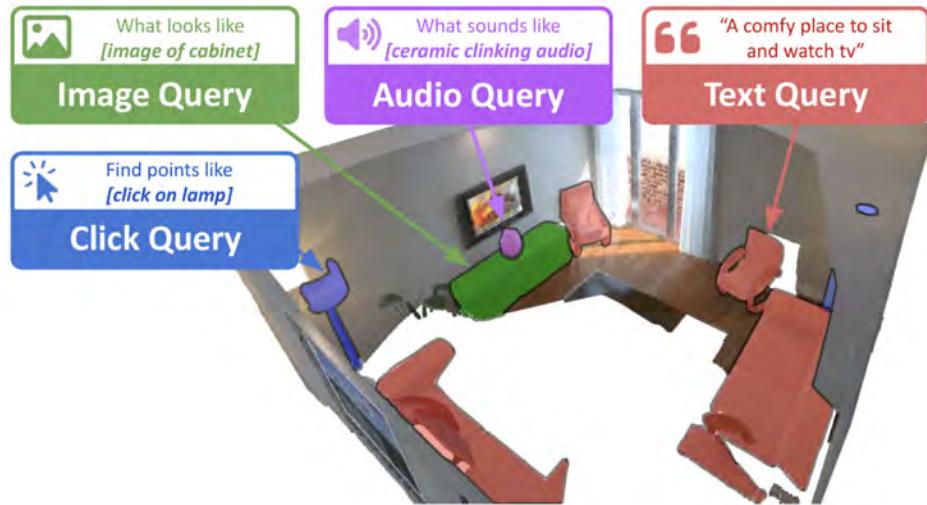


Figure 17.1 By embedding representations from multi-modal foundation models like CLIP [896] into SLAM maps, we can interact and query map elements in new ways. In this example, adapted from ConceptFusion[512], we have a 3D point cloud where points also store embeddings from other types of encoders. For example, when we query the map with a text query “A comfy place to sit and watch tv”, this text is passed through a text encoder and then we can look for all of the points in the map whose representation is sufficiently close (in terms of cosine distance) to the embedded query. The result is that we find all of the chairs and couches. The same methodology can be applied to any modality for which we have a multi-modal foundation model with aligned embedding spaces between the modalities. More details in Section 17.3.

## 17.1 Background and Terminology

### 17.1.1 A Brief History of Foundation Models

A *foundation model* is a machine learning model trained on a sufficiently vast *foundation model* dataset that we may reasonably expect should *generalize* to a wide variety of operating conditions. Training such models requires two key components: 1) deep learning architectures that are able to efficiently update a very large number of parameters, and 2) a very large (*e.g.*, internet-scale) dataset to train the model. The first model in computer vision that may fit into this definition (although not typically referred to as a foundation model) is AlexNet [606], which was a CNN model trained in a *supervised* fashion (meaning that the dataset contained labels

that were provided by human annotators) on the ImageNet database [260], which comprises 3.2 million labeled images. One issue with training such large models using traditional CNN architectures is that they suffer from the problem of *vanishing gradients*, where gradients that have to pass through many layers become very small and therefore training the parameters based on these gradients becomes very inefficient.

This was initially addressed through the introduction of residual networks [443], or ResNets, which introduce a direct pathway from one layer to the next, making training more stable and efficient. Vision encoders based on ResNets, which were pre-trained on ImageNet, formed the basis for the development of deep-learning-based *closed-vocabulary* visual perception systems, such as object detection models including YOLO [916], RCNN [384], and many others.

However, constructing a large manually labeled dataset such as ImageNet is an arduous task. In the field of natural language processing (NLP), it was possible to automate this process. This was demonstrated by *Bidirectional encoder representations from transformers (BERT)* [268], which uses a corpus of data to generate a task with associated labels in an automated way as a method for learning meaningful representations. In the case of text, this is done by simply removing a word from a sentence and then trying to predict the missing word. The set of all sentences in the English Wikipedia ( $\sim 2.5$  billion words) and BookCorpus ( $\sim 800$  million words) were two of the first datasets used in this manner. This process is referred to as *unsupervised learning*<sup>1</sup> since no human labels are needed.

BERT also showcased the benefits of a new type of model architecture called the *Transformer* [1120], which was based on the concept of *attention* [43]. Attention enabled capturing longer-term dependencies in the input data. Concurrently with BERT, the first *Generative pre-trained transformer (GPT)* model was released, which was also based on the transformer architecture and trained on the BookCorpus dataset and showed similar capabilities.

While it was relatively intuitive to perform this type of unsupervised training with language, it was not immediately clear how to extend this concept to higher-dimensional inputs such as images. Words contain specific semantic meanings in a way that random patches of an image may not. This problem began to be unlocked through the development of *contrastive* learning-based approaches combined with vision transformers [280].

#### *contrastive learning*

In *contrastive learning* approaches, such as SimCLR [185], representations are learned by augmenting the input image in some way (such as removing patches, flipping it, etc.) and then creating a loss function that pushes the original and augmented image closer in representation space (positives), while pushing all other

<sup>1</sup> It is fairly common in the literature to distinguish between *self-supervised* learning, where a label is manually generated, and *unsupervised* learning, where there is no explicit label, but we make no such distinction here and refer to all methods that do not require human annotations as *unsupervised*.

inputs or modified inputs away (negatives). Through this process, the hope is that the representations that are learned are invariant to the data augmentation and are therefore robust for downstream visual perception tasks (see Figure 17.2).

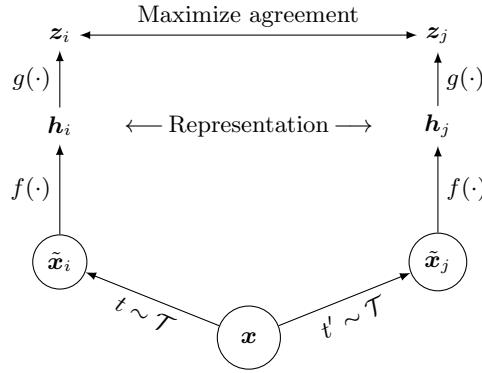


Figure 17.2 Reproduced from [185]. An input  $\mathbf{x}$  is read through two different data augmentation operators ( $t \sim T$  and  $t' \sim T'$ ) and applied to each data example to obtain two correlated views. These augmented inputs are then fed through an encoder network  $f(\cdot)$  to obtain a representation, and a projection head  $g(\cdot)$ , which is used for training through a contrastive loss.

Concurrently with the development of contrastive learning, the research community started applying transformers to the visual domain [280], as shown in Figure 17.3. By leveraging a similar idea of ‘tokenization’ of an image and multi-headed self-attention, the *vision transformer* was applied to classification and shown to be competitive with state-of-the-art supervised ResNet models on classification tasks.

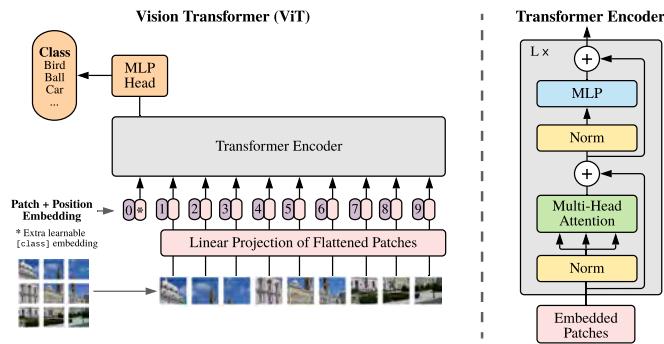


Figure 17.3 Reproduced from [280]. The vision transformer splits an image into fixed-size patches and linearly embeds each of them together with position embeddings. The resulting sequence of vectors is fed as input to a standard Transformer encoder. To perform classification, an extra learnable ‘classification token’ is added to the sequence.

The combination of these two approaches led to breakthroughs in unsupervised visual representation learning. One canonical example was *Deeper into neural networks (DINO)* [154], which used a teacher-student distillation setup (instead of contrastive learning) for unsupervised training of a vision transformer encoder, and was remarkably effective at learning visual representations.

Contrastive learning also enabled the alignment of representation spaces *across modalities*. For example, *CLIP* leverages image-text pairs collected from the internet and uses a contrastive loss to push an image and its associated text caption closer together in feature space, while pushing all other examples away [896].

At this point, the majority of the theoretical tooling was present to enable feature-based and generative foundation models (see Section 17.2 for further details about this distinction). Further improvements to models, datasets, and workflows, combined with ever-increasing computational capabilities, continue to improve the representational capabilities of these models, showing impressive ability to generalize to unseen tasks and settings. However, it is somewhat unclear whether this progress will continue at the same rate in the future.

As we turn our focus to the ways in which these models can be useful in the context of SLAM, it is important to keep in mind that the existence of truly massive, internet-scale datasets has been one of the key ingredients in the incredible representational power of these models. However, several data modalities that are commonly used in SLAM systems, and have been discussed in previous chapters, such as LiDAR or radar, among others, lack readily available data at this scale.

### 17.1.2 Terminology and Scope

We seek to make precise exactly what types of methods we consider to be in scope for the remainder of this chapter. A first concept to make precise is exactly what we mean by ‘open-world’, as this represents the key distinction between this chapter and Chapter 16.

*open-world queries*

**Open-World Queries.** There are (at least) two ways that we could envisage that this open-world concept could be applied in the context of SLAM. In the first paradigm, we could consider using open-world variants of front-end perception systems. Examples of such object detection systems include YOLO-World [194], which is an open-vocabulary variant of YOLO [916], and OV-DETR [1261], which is an open-vocabulary variant of the “Detection Transformer” (DETR) [146]. However, it is important to emphasize that while these models do admit open-vocabulary text-conditioned queries, this conditioning must happen *at the time that the perception model is queried*, which in the case of SLAM is at the time of map construction. Consequently, the resulting map should be considered to be *closed-world* (or closed set) since the decision about which concepts should be represented is enforced at the time of map creation.

A second paradigm typically embeds open-world representations into the SLAM

map in a way that enables *open-world queries* when the map is actually being used. We will focus primarily on this second paradigm in Section 17.3.

**Localization vs. Mapping vs. SLAM.** Foundation model representations have been used for visual place recognition (VPR) and localization/loop closure detection. For example, AnyLoc [546] studies which layers and features from DINO [154] can be aggregated to derive image descriptors for VPR under spatial, temporal, and viewpoint variation. While this is certainly an important line of work related to SLAM, we will focus more directly in this chapter on the data structures and algorithms that enable open-world querying and task specification in SLAM maps. The vast majority of the methods that we will discuss in the remainder of the chapter are actually pure *mapping* algorithms since they assume the availability of *posed* camera frames (for example from some upstream SLAM system such as the ones we discussed in earlier chapters). As such, in most cases, the problem reduces to the question of *how to embed foundation model representations into SLAM maps?*

In the remainder of the chapter, we first continue with a more detailed technical discussion of some of the most influential and useful foundation models for open-world mapping (Section 17.2). Then, we present several examples of how these open-world foundation model representations are being incorporated and embedded into SLAM maps to enable joint geometric/semantic reasoning (Section 17.3). Finally, we look forward to what might be next in this rapidly evolving field (Section 17.4).

## 17.2 Foundation Models for Spatial AI

In this section, we introduce three classes of foundation models helpful for building open-world map representations. We first distinguish between **feature-based** and **generative** foundation models. Feature-based foundation models such as CLIP [896] extract real-valued vectors from text or images (*i.e.*, features) and rely on feature space computations to achieve tasks (*e.g.*, compute some distance between two feature vectors to measure similarity between the corresponding text or image). In contrast, generative foundation models can generate novel data given text and/or image prompts. The distinction is not always clear-cut: some of the feature-based models we will introduce are actually used as sub-components in generative models, and conversely, features can technically be extracted from generative models. However, differences in outputs and typical use cases in the mapping context support the distinction. Generally speaking, we also note that inference with generative models, while potentially more powerful, tends to be more computationally expensive than extracting features. We conclude this section with a brief overview of useful computer vision models for **class-agnostic image segmentation** since these models are very widely used in the various approaches that we will cover in Section 17.3.

### 17.2.1 Feature-Based Foundation Models

Features, also known as embeddings, representations, or codes, are real-valued vectors resulting from the encoding of an input data point by a deep neural network. We will only briefly mention network architectures and training objectives and simply denote different feature extractors as functions, leaving the reader to consult works such as [392] for a proper overview of deep learning. For the purposes of this chapter, we will consider that we have an encoder  $f_X$  (typically a deep neural network) that can be applied to an input  $x$  of modality  $X$  resulting in  $d$ -dimensional features  $f_x$  according to  $f_x = f_X(x) \in \mathbb{R}^d$ .

The field of deep learning aims to learn such encoding functions with useful properties. A key property for this chapter will be the ability to easily measure meaningful *similarities* between data points. Distances in feature space will help us answer queries such as “*How similar are these two text excerpts?*” or “*How similar is this text caption to this image?*”

Research in self-supervised learning objectives and deep architectures has led to the emergence of feature extractors that can act as a foundation for a wide range of applications. Modern feature extractors are pre-trained on large internet-scale NLP and computer vision datasets. While it is possible to fine-tune these models for specific tasks or even use their features as inputs to downstream classifiers or regressors, the mapping methods we consider in this chapter mostly use the features “as is”. Here, we present a brief overview of some available text, vision, and multi-modal feature extractors.

**Natural Language.** BERT [268] transformed the NLP landscape and introduced bi-directional transformers pre-trained on a masked language modeling objective. The resulting model achieves strong performance on a number of downstream NLP tasks, with and without fine-tuning. While BERT provides features at the level of tokens (*i.e.*, sub-strings of the input), SentenceBERT [921] adds a pooling operation and fine-tunes BERT to output sentence-level features that can efficiently be compared with cosine similarities (the cosine of the angle between the two embeddings treated as vectors originating from the origin). SentenceBERT model checkpoints are available in a number of languages and have been fine-tuned for a variety of NLP tasks.

**Vision.** DINO [154] trains vision transformers [280] with a self-supervised learning scheme involving a student network matching the predictions of a teacher network. A key training ingredient is multi-crop augmentation: the teacher is exposed to global views ( $> 50\%$  of the image) while the student has to process both local views ( $< 50\%$  of the image) and global views of the same image, encouraging “local-to-global” learning. DINoV2 [828] proposes various improvements to the training procedure and introduces a curated dataset of 142M images. The result is a powerful visual feature extractor with a strong holistic understanding of images. DINO outputs *image-level* visual features that generalize zero-shot to downstream

tasks such as image classification or image retrieval. For example, distances in the DINO feature space can serve as a foundation to build k-nearest neighbors classifiers for specific supervised tasks. Even simple models such as logistic regression can achieve strong performance in the image domain when trained on DINO features. The transformer architecture underpinning DINO also allows for the prediction of *patch-level features* (*e.g.*, a feature vector for every 8x8 patch in the image). Training additional models on these dense features can yield strong performance on pixel-resolution downstream tasks such as semantic segmentation, depth estimation, and surface normal estimation [828, 45]. Directly using distances in feature space has also been explored for multi-view correspondence estimation [45].

**Multimodal.** A learned feature space can be a powerful tool to reason about similarities between data points from the same modality. Different modality-specific encoders can also map to a common *aligned feature space* to enable multimodal capabilities. CLIP [896] is a notable example of this approach in the vision-language setting, training an image encoder  $f_{\text{image}}$  and a text encoder  $f_{\text{text}}$  (Figure 17.4). Given an RGB image  $\mathbf{I}$  and a text query  $T$ , CLIP extracts unit-norm features  $\mathbf{f}_I = f_{\text{image}}(\mathbf{I})$  and  $\mathbf{f}_T = f_{\text{text}}(T)$ . This enables image-text comparisons using the cosine similarity  $\langle \mathbf{f}_I, \mathbf{f}_T \rangle$ .

CLIP encoders are trained using a contrastive learning scheme on 400M image-caption pairs collected from the internet, yielding features that encode a broad array of visual and textual concepts out-of-the-box. CLIP features can be repurposed for a variety of downstream tasks without any fine-tuning or retraining. For example, we can construct a *zero-shot image classifier* by defining a set  $C$  of class labels in text form, possibly using a template sentence (*e.g.*, {a picture of a dog, a picture of a cat}), and use CLIP similarities to predict a class for an image  $\mathbf{I}$  with

$$\arg \max_{c \in C} \langle \mathbf{f}_I, \mathbf{f}_c \rangle. \quad (17.1)$$

Reformulating a classifier for a new problem is simply a matter of specifying a new class set through the definition of the text prompts, allowing transfer to a wide array of tasks. The classifier in (17.1) can be extended to other modalities such as sound [421], given an appropriate encoder.

Extracting dense patch-level image features from CLIP vision encoders is also possible. However, one should not expect these features to be language-aligned, and some studies have found them to be more limited for vision and 3D-related tasks in comparison to models such as DINO [828, 45].

### 17.2.2 Generative Foundation Models

Large language models (LLMs) are transformers trained for text generation, generally using self-supervised objectives. The models are typically large (*i.e.*, many

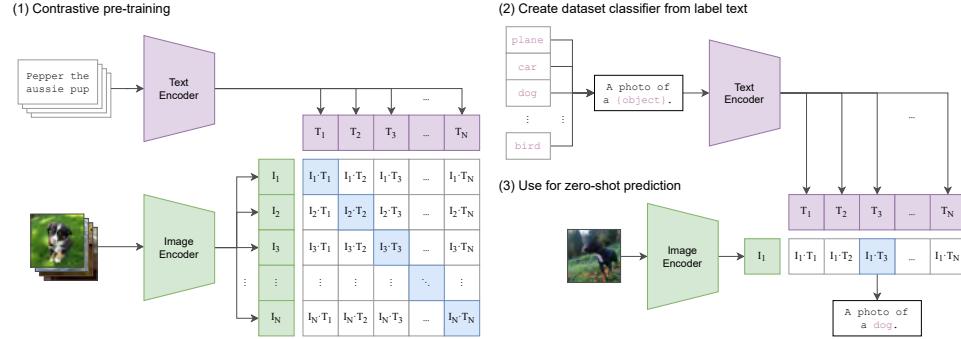


Figure 17.4 Figure reproduced from [896]. (1) CLIP learns image and text features with contrastive learning. Image and caption features from the same image-caption pair are incentivized to be close in feature space (positives, in blue) while image features and features from other randomly sampled captions are pushed away (negatives, in white). We note that the pre-training of CLIP does not rely on the definition of any particular class set. (2) After training, class text labels (possibly in a template) can be encoded to define a classifier. (3) The feature vector extracted from an image can be compared with class features for zero-shot prediction.

learnable parameters) and trained on a massive text corpus, resulting in impressive generalization. Here we outline key aspects of LLM training and inference and refer the reader to [1120, 1110, 1101] for an in-depth treatment of transformers in the NLP context.

The first step in language modeling is tokenization. Tokenization is the process of segmenting the input text into tokens (*e.g.*, words, subwords, characters, or bytes) defined in a vocabulary. Each token is assigned to a unique integer value and has a corresponding learnable embedding. LLMs will process text as a sequence of token embeddings.

#### next-token prediction

The specifics of LLM training are beyond the scope of this chapter, but the core learning objective is *next-token prediction*. Given a context of  $T - 1$  tokens  $w_{1:T-1}$ , LLMs learn to predict the probabilities  $p(w_T|w_{1:T-1})$  of the next token. At inference time, LLMs will accept an initial context of text instructions (a prompt) and autoregressively generate text by predicting the next token, adding it to the context, and repeating the process until some stopping condition is met.

LLMs can, in principle, be used for any task where instructions, inputs, and outputs can be specified in text form. Examples include interactive chatbots, code generation, text summarization, or symbolic planning. Prompt design will significantly impact task performance and has driven research on prompt engineering techniques such as chain-of-thought (prompting the model to think step-by-step) and in-context learning (providing input-output examples in the prompt). Some widely used LLMs include the LLaMa family of models [1101] and the GPT mod-

els [5]. Many others are available with newer, more powerful models being released frequently.

There is great interest in extending the general abilities of LLMs to other modalities, such as vision. Vision language models (VLMs) share most of the components of LLMs in addition to accepting image tokens as part of their input. LLaVA [646] achieves multimodal capabilities by connecting the CLIP vision encoder to an LLM and training on vision and language data. Recent GPT models also support vision inputs and outputs. Given an adequate prompt, VLMs can perform tasks such as image classification, image captioning, and more general visual question answering.

### 17.2.3 Class-Agnostic Image Segmentation

Image segmentation is a key component in many mapping methods and will be essential for reasoning about the semantics of specific image regions. We will be interested in detecting a set of  $R$  segmentation masks  $\{\mathbf{Z}_i \in \{0, 1\}^{H \times W}\}_{i=0}^R$  representing the likely objects in an RGB image  $I$  of width  $W$  and height  $H$ . Critically, we need the mask proposals to be class-agnostic to properly capture the breadth of objects typically found in robot environments (and to exploit the open-vocabulary nature of the foundation models that we just introduced). Throughout this chapter, we will denote this function as  $\text{Seg}(\cdot)$ .

The main tool to implement this function is Segment Anything Model (SAM) [580, 908]. SAM is based on large-scale supervised learning for class-agnostic image segmentation. The result is a promptable segmentation model that takes as input pixel coordinates, a bounding box, or even another segmentation mask and predicts a segmentation mask. SAM can also automatically process a full image to generate a set of masks describing different instances in the image. While SAM can be applied directly as a standalone segmentation model, it can also process the bounding box detections from an open-vocabulary object detector [678] to achieve better object definition and reduced inference time.

## 17.3 Open-World Mapping

In Chapter 5, we saw how geometric quantities such as occupancy or distance fields could be stored in maps, and Chapter 16 extended this to semantics, primarily through the storage of additional class information. While certainly offering a reasonable discrete approximation of scene semantics, the class framework faces a number of limitations in practice. Chief among them is the *closed-set* assumption: classical semantic maps require *a priori* knowledge of the vocabulary of classes we wish the map to support. For example, any mapping system relying on image classifiers or object detectors inherently assumes that the world can be fully described by the vocabulary of their perception front-end.

In practice, predefined object classes only offer a very coarse approximation of

*closed-set semantics*

reality. Consider the typical objects found in a toolbox. Most classifiers will lack the vocabulary to fully describe the usual objects in our box. While “tool” might be an acceptable general description for each object, more specific labels such as “screwdriver” or “hammer” are certainly more desirable. Even then, we might be interested in additional object properties such as the color of specific tool instances (“red hammer”, “blue screwdriver”), their material (“metal”, “rubber”) and potentially other physical properties (“hard”, “soft”, “hot”, “cold”) that are typically lost if we assign a single class to a map element. Of particular interest to robots, we might want to encode additional information on the typical uses and affordances of a given object (“cutting”, “measuring”). In the case of objects with depictions (“a painting of Ronaldo”), an ideal map would encode both the medium (“painting”) and the content, even in the case of a specific cultural reference (“Ronaldo”). Meeting all these objectives with a predefined set of classes is an arduous task. A vocabulary containing hundreds of classes may still fall short at some level, even on common objects, and the issue is magnified for rare “long-tail” objects that are unlikely to be included in the vocabulary in the first place. Other continuous properties, such as the typical sound an object can make, are inherently hard to represent with a discrete class label.

To remove the closed-set assumption and enable support for a broader range of concepts, we can replace class labels in the map with some high-dimensional continuous features extracted by the foundation models we introduced in Section 17.2. The large datasets used to train these feature extractors ensure that the features can represent a vast, *open* set of objects, along with their associated properties. While grounded features themselves are not directly usable, their multimodal alignment allows for comparison with queries expressed in different modalities, primarily through natural language, and enables performing a number of tasks in a zero-shot manner.

Throughout this section, the primary sensor of interest will be RGB-D cameras, which support both RGB feature extraction using image-based foundation models and pixel-level conversion to 3D representations like point clouds. We generally assume a sequence of *posed* RGB-D frames in a static scene, therefore focusing on the mapping problem as opposed to the full SLAM problem. We process each frame sequentially to incrementally build our maps.

*closed-set semantics*

*posed frames*

*dense  
maps*

*open-vocabulary*

### 17.3.1 Dense Representations

In this section, we study the three main components involved in building *dense open-vocabulary maps*: 1) the choice of 3D representation, 2) the extraction of open-vocabulary features from sensor data using foundation models, and finally 3) the aggregation across frames and the spatial grounding of these features. Finally, we discuss the process of performing inference or querying the map.

**Map Representation.** Open-vocabulary mapping embeds semantic features from foundation models into one of the dense spatial representations introduced in Chapter 5. Without loss of generality, we will assume that a map representation is comprised of a set of map elements  $M \triangleq \{m_k\}$  where each map element  $m_k \triangleq \{\mathbf{p}_k, \mathbf{f}_k\}$  is composed of a geometric component  $\mathbf{p}_k$  (voxel location, 3D point, mesh face etc.), and a semantic component or feature vector  $\mathbf{f}_k$ .

The choice of representation is mainly driven by the downstream application: navigation methods will favor cell-level feature aggregation in a 2D grid [484] while a mobile manipulation pipeline may opt for voxel-grids [675]. ConceptFusion [512] and OpenScenes [860] model space and semantics in a map with an unordered set of 3D points. Open-Fusion [1210] utilizes a volumetric representation with a TSDF. A hybrid approach —taken by VLMaps [484]— is to accumulate the information in a 3D point cloud but then project it into a 2D grid map.

It should be emphasized that the geometric component of the map elements,  $\mathbf{p}_k$ , is assumed to be provided by an upstream SLAM system, such as the ones that we have seen in previous chapters.

**Feature Extraction.** We turn our attention now to the choice of which features to embed in the 3D representation and how to obtain them. A common approach is to extract intermediate semantic features from the RGB data of a given frame  $\mathcal{I}$  using a foundation model  $f$ , *e.g.*, one of the models described in Section 17.2.1. One issue is that foundation models typically yield image-level features encoding information on the whole collection of objects in the camera frustum. While this larger context is valuable, we are also interested in more fine-grained, local semantics to properly describe the different parts of the image, potentially at a pixel-level resolution. But there is a question of how to obtain these more fine-grained features.

One approach is to use a foundation model that directly outputs pixel-level [645] or region-level [379] features. These models are trained by fine-tuning a foundation model like CLIP on a segmentation dataset. OpenScenes and VLMaps directly extract CLIP features from the RGB images through LSeg [645]. Open-Fusion [1210] adopts a different strategy by employing SEEM [1309] to directly extract region-based features, aiming for better computational efficiency.

However, recent evidence suggests that these dense features do not capture as many long-tail concepts as the original unaltered CLIP features [512]. An alternative approach taken in ConceptFusion aims to preserve the original CLIP features and relies on feature arithmetic to construct pixel-level feature vectors. However, this requires some heuristics to balance the need for local and global semantics information. The method balances the need for local and global semantics by computing:

- **Global features**  $\mathbf{f}^G$  by encoding the entire image with  $\mathbf{f}^G = f(\mathcal{I})$ ;
- **Local features**  $\mathbf{f}_i^L$  by processing all the masks  $R = \text{Seg}(\mathcal{I})$  inferred by a class-agnostic segmentation method (such as SAM [580]—see Section 17.2). Each mask

is converted to a bounding box to create tight image crop  $\mathbf{I}_i$ . Applying  $f$  results in mask-level features  $\mathbf{f}_i^L = f(\mathbf{I}_i)$ ;

- **Pixel-level mask features**  $\mathbf{f}_i^P = w_i \mathbf{f}^G + (1 - w_i) \mathbf{f}_i^L$ . The weights  $w_i \in [0, 1]$  for the linear combination are computed to emphasize local features that are (i) distinct from the global features and (ii) unique when compared to the other local features; and
- **Pixel-level features**  $\mathbf{f}_{u,v}^P$  by summing all the  $\mathbf{f}_i^L$  such that  $\mathbf{Z}_i[u, v] = 1$  (*i.e.*, the pixel-level features of the masks covering the pixel) and unit-normalizing the result.

Certainly, other methods of local and global feature merging could be possible, and performing this merge in a more principled manner could be an interesting avenue for future work.

In summary, assuming a pinhole camera model and pixel-feature correspondences, we can follow the procedure introduced in Section 5.1 to generate an open-vocabulary semantically enhanced organized point-cloud observation. Each point has a corresponding semantic feature vector calculated using one of the methods above (*i.e.*, either querying a pixel-resolution model or performing global/local feature merging). We refer to these as *range-feature* observations, analogous to the *range-category* observations introduced in Chapter 16.

**Feature Aggregation and Grounding.** Given a choice of spatial representation and a method for generating *range-feature* observations, the next component deals with how the features from different RGB-D frames that correspond to the same 3D location get combined into one that *annotates* the corresponding map element.

A straightforward approach is adopted by VLMaps where the *range-feature* observations in 3D from all frames are averaged if they correspond to the same 2D cell projection [484].

ConceptFusion [512] converts the *range-feature* observation to a global coordinate frame, then, for every element  $m_k$  in the map, all pixel-level features corresponding to the point  $\mathbf{p}_k$  are weighted and averaged by the feature confidences. The features,  $\mathbf{f}_k$  and confidences,  $c_k$  are updated using an averaging scheme:

$$\mathbf{f}_k \leftarrow \frac{c_k \mathbf{f}_k + \alpha \mathbf{f}_{u,v}^P}{c_k + \alpha} \quad (17.2)$$

$$c_k \leftarrow c_k + \alpha \quad (17.3)$$

where  $\alpha = e^{-\gamma^2/2\sigma^2}$  is the confidence assigned to each pixel-feature associated to the point being aggregated,  $\gamma$  is the radial distance to the camera center, and  $\sigma$  is a scaling term. Intuitively,  $\alpha$  assigns higher confidence to features closer to the camera center, similar to previous 3D reconstruction work [548].

Similarly, OpenScene [860] averages all the per-frame 2D features. However it also distills them into 3D features. The fusion of averaged 2D features and the distilled

3D ones is carried out through an ensemble mechanism. Open-Fusion [1210] takes a different approach and maintains in each occupied voxel only a key to a dictionary of features. The decision on which key to assign to the voxel results from a temporal feature matching.

Other works have investigated alternative feature aggregation or ensembling strategies to (i) avoid averaging high-dimensional feature vectors, and (ii) increase robustness to outlier feature vectors. For instance, the work [860] accumulates a feature set  $\mathcal{F}_k = \{\mathbf{f}_{k,1}, \dots, \mathbf{f}_{k,t}\}$  over time for all map elements and retains the feature vector with the lowest Euclidean distance to the other features in the buffer (the medoid vector). Clustering methods can reduce  $\mathcal{F}_k$  to a few representative cluster centroids [701] or make feature selection more robust by selecting the feature vector in  $\mathcal{F}_k$  that is closest to the centroid of a majority cluster [1179].

**Inference.** Once the features are grounded and aggregated in the map, they can be used to perform various tasks by comparing them to other features extracted from different modalities. For example, in the case of vision and language feature-alignment, as is the case with CLIP, *zero-shot semantic segmentation* can be achieved by encoding the textual class labels in a prompt list  $C$ . If the map contains map elements  $m_k$  with corresponding features  $\mathbf{f}_k$ , then point-level labels  $y_k$  can be predicted with

$$y_k = \arg \max_{C \in \mathcal{C}} \langle \mathbf{f}_k, \mathbf{f}_C \rangle \quad (17.4)$$

where  $\mathbf{f}_C$  is the feature representation obtained by encoding the prompt  $C$  with the text encoder:  $\mathbf{f}_C = f_{\text{text}}(C)$ . Note that this mirrors exactly the image classifier discussed in Section 17.2. Critically,  $C$  does not need to be known when building the map and can be redefined at will to predict different labels. The segmentation process for other map representations, such as grid maps and voxel maps, is analogous.

However, the main use case of open-vocabulary maps is arguably *querying*. Open-ended querying is achieved by encoding a query and comparing query features with all map features. In the CLIP context, one can encode a free-form text query  $T$  to extract the corresponding feature vector  $\mathbf{f}_T = f_{\text{text}}(T)$  and score the points in the map  $M$  based on the query similarities  $\langle \mathbf{f}_k, \mathbf{f}_T \rangle$ .

The resulting similarities can be thresholded to return a set of relevant map elements. The spatial coordinates of the returned elements can also be clustered to identify different instances. Again, no prior knowledge of the queries is required at map building time and the generality of the multi-model foundation model will ensure that even the most specific queries can be reasonably answered.

While language is a natural way of specifying queries, any modality with an aligned encoder can be used to extract query features. Query features can also be

retrieved through user interaction. As an example, in addition to text, ConceptFusion explores a range of *multimodal queries* (see Figure 17.1):

- **Click query** is taken as the feature vector  $\mathbf{f}_k$  of a clicked point in a visualization of the map  $M$ ;
- **Image query**  $\mathbf{f}_I = f_{\text{image}}(\mathbf{I}_{\text{query}})$  using the image encoder  $f_{\text{image}}$  on a query image  $\mathbf{I}_{\text{query}}$ ;
- **Audio query**  $\mathbf{f}_s = f_{\text{audio}}(s)$  using the AudioCLIP [421] sound encoder  $f_{\text{audio}}$  on a sound clip  $s$ .

Zero-shot inference is more limited when using vision-only features (*e.g.*, DINO [154]). Such maps will still support image queries, local image queries (*e.g.*, with image clicks), and map click queries. Users can click on a few objects in some reference images to achieve accurate segmentation of objects and parts in the map [742].

### 17.3.2 Object Maps and 3D Scene Graphs

While dense open-vocabulary maps can model semantics at a very granular level, they scale poorly with the size of the environment due to the high dimensionality of commonly used feature vectors. In our ConceptFusion example, we saw the value of considering masks in images to extract local features. This strategy leverages spatial correlations: nearby pixels in an image tend to belong to the same objects and share the same semantics. This observation also applies to maps in general. We expect nearby map elements (points, voxels) to generally share similar semantics, which suggests that semantic features could be stored at the level of objects without incurring a large loss of information.

*open-vocabulary maps*

*object maps* In this section, we study the construction of *open-vocabulary object maps* and 3D scene graphs. Building on the maps introduced in Section 5.1, we model every object in the scene with a geometric representation and only store one feature vector per object, drastically reducing the memory footprint of the map. Object-centric representations are also particularly well-suited for embodied AI applications since they lend themselves more naturally to object retrieval queries and object-centric task planning. We first describe how to build an open-vocabulary object-centric map using steps from OVIR-3D [701] and ConceptGraphs [407] as examples. We then discuss different ways to model relationships between objects (object scene graphs) and how to integrate objects in spatial hierarchies (hierarchical scene graphs) and highlight HOV-SG [1179] as an example of such an approach.

**Map Representation.** We represent the map as a set of  $J$  objects  $V_O = \{o_j\}_{j=1}^J$ . Each object  $o_j$  is characterized by a geometric representation and a feature vector. The geometric representation can range from sparse abstractions such as object centers [178] and 3D bounding boxes to dense representations like 3D point clouds [407, 1179, 1063]. For the examples in this section, we will describe each object with a 3D point cloud  $P_{o_j}$ , a feature vector  $\mathbf{f}_{o_j}$ , and a detection count

$n_{o_j}$ . We incrementally build the map by processing incoming posed RGB-D frames, adding or initializing objects as needed.

**Feature Extraction.** Object-centric maps typically rely on a class-agnostic segmentation method (such as SAM [580]) to extract a set of masks  $\mathbf{R} = \text{Seg}(\mathbf{I})$  from the current image. We convert each mask to a bounding box, create the corresponding image crop  $\mathbf{I}_i$  and extract some mask-level features  $\mathbf{f}_i^M = f(\mathbf{I}_i)$ . This procedure is identical to the extraction of local features in ConceptFusion.

Using the depth data, camera pose and camera intrinsics, we backproject the pixels of every mask to 3D resulting in mask point clouds  $\mathbf{P}_i^M$  which are converted to the map frame. Mask point clouds can also be denoised at this stage (*e.g.*, with DBSCAN [311]). In tandem with the mask features, we obtain a *mask-feature* observation  $\{(\mathbf{P}_i^M, \mathbf{f}_i^M)\}_{i=0}^R$ .

**Object Association.** We now need to measure the similarity of our masks with existing objects in  $V_O$ . Methods will usually rely on some notion of geometric and semantic similarities. Geometric similarity can be measured as the nearest neighbor ratio  $\phi_{geo}(\mathbf{P}_i^M, \mathbf{P}_{o_j}) \in [0, 1]$  defined as the proportion of points in point cloud  $\mathbf{P}_i^M$  that have nearest neighbors in point cloud  $\mathbf{P}_{o_j}$ , within a distance threshold of  $\delta_{nn}$  [407, 1179]. Semantic similarity can be measured using a function of feature similarity such as  $\phi_{sim}(\mathbf{f}_i, \mathbf{f}_{o_j}) = 1/2(\langle \mathbf{f}_i, \mathbf{f}_{o_j} \rangle + 1) \in [0, 1]$ .

Geometric and semantic similarities can be combined in an overall score or thresholded separately to identify potential mask-object matches. A greedy strategy [407] can associate incoming masks to a single object in  $V_O$ . Other more sophisticated schemes allow one mask to be associated with multiple objects [701, 1179]. If a mask has no match, its geometry and features can be used to initialize a new object in  $V_O$ .

**Object Fusion and Feature Aggregation.** In the event of a match between  $o_j$  and the  $i^{th}$  mask, object  $o_j$  needs to be updated. One approach is to combine point clouds and average features:

$$\begin{aligned}\mathbf{f}_{o_j} &\leftarrow \frac{n_{o_j} \mathbf{f}_{o_j} + \mathbf{f}_i}{n_{o_j} + 1} \\ \mathbf{P}_{o_j} &\leftarrow \mathbf{P}_{o_j} \cup \mathbf{P}_i^M \\ n_{o_j} &\leftarrow n_{o_j} + 1\end{aligned}$$

with  $\mathbf{P}_{o_j}$  being subsequently down-sampled to remove redundant points. The alternative feature aggregation strategies discussed for dense maps also apply here.

Open-vocabulary methods will also rely on additional processing steps to refine object estimates. Methods may periodically (*i.e.*, every few frames) run association and fusion algorithms between the objects in  $V_O$  to minimize the number of redundant objects [407, 701]. It is also possible to remove objects with a low detection count  $n_{o_j}$  either periodically during mapping or after processing all frames.

**3D Segmentation.** An alternative to 2D image segmentation and mask asso-

ciation would be to accumulate all frames in a global scene point cloud and then directly perform object segmentation in 3D, using either a learned model [1063] or geometric techniques such as region growing [538]. By reprojecting 3D object masks, methods can identify representative images for each object and extract features accordingly. 3D segmentation approaches have the potential to be faster since they avoid segmenting individual RGB frames and associating their segments. However, they do not lend themselves naturally to incremental mapping since the full scene point cloud is required.

**Inference.** The zero-shot segmentation and querying mechanics are very similar to the dense open-vocabulary mapping scenario. For zero-shot segmentation, 3D points inherit the predicted class label of the object to which they belong. Assuming a point  $p_k \in P_{o_j}$  belonging to an object with some CLIP-aligned object features  $f_{o_j}$ , the predicted point-level class label is defined as

$$y_k = \arg \max_{C \in C} \langle f_{o_j}, f_C \rangle$$

where  $C$  is a set of classes in text form and  $f_C$  is the CLIP text feature vector of a given class prompt. Objects and their similarities can also be adapted for instance segmentation [1063].

Queries become significantly more useful as they can now be used for object retrieval. Given some text query,  $T$ , its features  $f_T$ , we can retrieve the most relevant object in the object set  $V_O$ :

$$o^* = \arg \max_{o \in V_O} \langle f_o, f_T \rangle$$

Object retrieval capabilities are extremely useful for language-driven goal specification in the context of navigation and manipulation. For instance, a query such as “Something to drink” will correlate strongly with the object features of a soda can. The corresponding point cloud can be forwarded to a motion planning pipeline to navigate to, inspect, and potentially manipulate the soda can.

**Language Descriptions.** Language-aligned object features work best when compared to the language features of short descriptive queries. They typically fail to capture the nuances of more complex queries involving negation or affordance information. “Something to drink other than a soda can” will likely correlate with a soda can features regardless of the query intent. Moreover, features are not *explicit* and cannot be directly understood by human users or LLMs.

An alternative to object features is to directly use free-form language to describe objects in the map. Language as an alternative representation of semantics is made possible thanks to recent progress in developing VLMs that can reasonably describe most objects. A simple way to extract object descriptions is to track the best image crops of each object (*e.g.*, by tracking the masks that contributed the most points

to an object) and pass them to a VLM along with a prompt such as “**describe the central object in the image**”. Individual object views can be captioned before obtaining on final summary per object using an LLM. Some VLMs also directly accept multiple images as input. We note that some methods use both features and language descriptions [407, 164].

Object captions are much more expressive than closed-set class labels and benefit from the general object understanding of VLMs. For querying, an LLM can be instructed to “search” for the most relevant object in the map given the list of object captions and a user query. This LLM-based object retrieval strategy has been found to address some of the limitations of feature-based object retrieval with more complex object queries [407]. We note, however, that LLM captioning and inference are typically much slower than feature-based alternatives. Alternative text-based representations such as language tags have also been leveraged effectively [1272].

**Object Scene Graphs.** We can consider a more general map representation  $G = (V_O, E_O)$  where the edge set  $E_O$  explicitly models the pairwise relationship between the different objects in  $V_O$ , forming a 3D scene graph. Much like object semantics, we aim to infer open-vocabulary edge descriptions either using language or features. ConceptGraphs [407] relies on object captions and positions to infer spatial relationships such as “**a on b**” or “**b in a**” with an LLM, and discusses the extension of spatial edge types to other relationships a language model can interpret, such as “**a backpack may be stored in a closet**” and “**sheets of paper may be recycled in a trash can**”. Edge features can be derived by encoding images where two objects are co-visible with a foundation model [587] and there is a growing interest in learning relationships using graph neural networks [586, 588]. Querying in the context of object scene graphs requires additional thought: OVSG [164] maps contextual language queries to graph structures using LLMs and proposes a graph matching approach to compare them with the scene graph [164].

**Hierarchical Scene Graphs.** Orthogonal to the object-object relations discussed before, multiple works have investigated the partitioning of hierarchical scenes while relying on open-vocabulary semantics [1179, 468, 1029]. In this case, following the definition of hierarchical graphs introduced in Section 16.4, the observed environment is sparsely modeled as a graph  $G = (V, E)$  that is hierarchically partitioned into  $l$  ordered concept layers. This yields a set of nodes  $V = \cup_{i \in 1:l} V_i$ , which can come in various forms depending on the target environment. Along this research line, HOV-SG [1179] separates indoor environments into floors, regions, objects, and Voronoi nodes covering navigable free space, while Clio [720] models regions, places, and objects. Other works adapt to typical urban outdoor environments that are decomposed into, *e.g.*, roads, lanes, and static or dynamic objects [1029, 263].

In the following, we discuss typical indoor scenes and follow the approach introduced by HOV-SG. We assume the following map factorization employing pair-wise

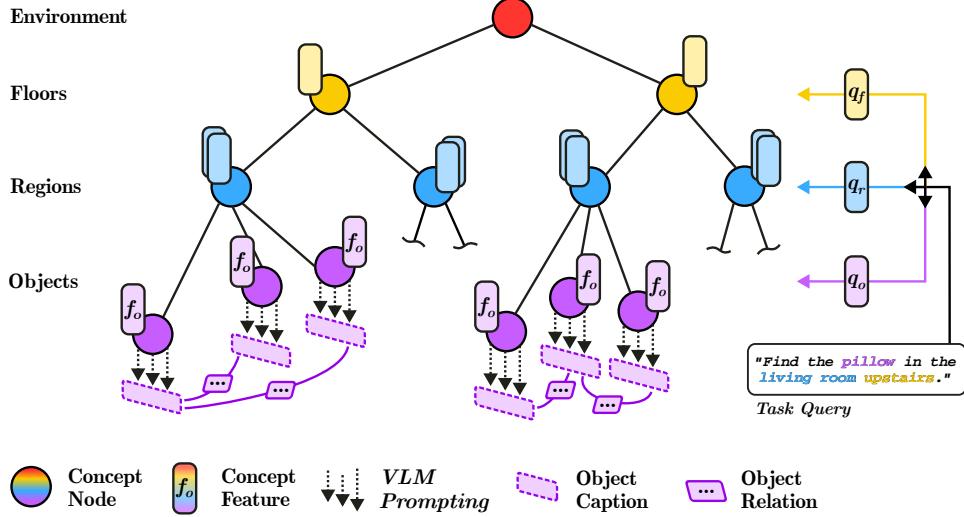


Figure 17.5 Open-vocabulary 3D scene graphs including hierarchical environment partitioning [1179] (HOV-SG) and language-level object-object relations [407] (Concept-Graphs).

disjoint sets of nodes and edges, respectively:

$$\mathcal{V} = \mathcal{V}_E \cup \mathcal{V}_F \cup \mathcal{V}_R \cup \mathcal{V}_O \quad \mathcal{E} = \mathcal{E}_{EF} \cup \mathcal{E}_{FR} \cup \mathcal{E}_{RO}, \quad (17.5)$$

where  $\mathcal{V}_E$  represents a single root node and  $\mathcal{V}_F, \mathcal{V}_R, \mathcal{V}_O$  denote the floor, region, and object nodes. Thus, we extend the prior object factorization of ConceptGraphs [407] to multiple conceptual layers. Accordingly, the set of edges  $\mathcal{E}_{EF}$  connects floors with the root node,  $\mathcal{E}_{FR}$  floors with regions, and  $\mathcal{E}_{RO}$  regions with objects, respectively. This layering enables hierarchical relation identification, smaller memory footprints, and fast graph search. HOV-SG supports language grounding of all concepts, ranging from floors to objects, by equipping each conceptual node with at least one open-vocabulary concept feature that is attained using CLIP. This is visualized in Figure 17.5 and detailed in the following.

Similar to the setting of ConceptGraphs, HOV-SG performs incremental RGB-D mapping and class-agnostic instance prediction while assuming access to accurate camera poses. This produces a point cloud that is separated into multiple floors based on binning observed points along the gravity-aligned height axis of the scene and subsequent peak identification. In order to refer to the obtained floors via language, each floor gets assigned a templated CLIP text feature of the form “*floor {X}*” or “*upstairs*”.

Next, we derive region estimates that satisfy geometric constraints and exhibit semantic coherence. Downstream tasks involving navigation require region primitives that adhere to spatial continuity and structural boundaries represented by,

*e.g.*, walls or doors [468]. In contrast, semantic higher-level reasoning policies favor semantically consistent grounding of regions. A typical corner case of this is a combined kitchen and living room area covering large extents but sharing a significant number of semantically distinct sub-regions. While HOV-SG employs a pure geometric approach at region segmentation by thresholding Euclidean distance fields per floor, other approaches cluster navigational nodes obtained through a Generalized Voronoi Diagram (GVD) given a task instruction [720], or harness doors as logical indicators of region boundaries [468].

To enrich region concept nodes with open-vocabulary features, we obtain the camera views of all camera poses falling into the interior set of a region segment, respectively. For each assigned camera view, we obtain the corresponding image-level CLIP embedding. In order to represent a distinct region, we distill the set of camera views into  $K$  representatives  $\{\mathbf{f}_{r,k}\}_{k=1}^K$  using k-means clustering. We identify region categories by scoring those representatives against a putative set of candidate region categories  $C_R$  that are encoded via CLIP, yielding  $\{\mathbf{f}_{R,c}\}_{c=1}^{|C_R|}$ . A typical set could include the following regions categories: {“kitchen”, “bathroom”, “living room”, “corridor”}. We obtain category votes for each representative  $\mathbf{f}_{r_k}$  through dot-product scoring:

$$c_{r_k}^* = \arg \max_{c \in C_R} \langle \mathbf{f}_{r_k}, \mathbf{f}_{R,c} \rangle, \quad (17.6)$$

where  $c_{r_k}^*$  represents the highest-scoring region category per region representative  $\mathbf{f}_{r_k}$ . By computing the majority vote among all representatives, a single region category is identified.

Furthermore, HOV-SG demonstrates natural language-based navigation within large-scale environments by hierarchically scoring against concept layers. Given a complex text query such as “*go to the plant in the living room on the first floor*”, a generative foundation model (GPT) decomposes the long query into multiple distinct captions based on the concepts imposed during the mapping stage. In turn, those concepts (“plant”, “living room”, “first floor”) are encoded to match the CLIP feature space of the constructed scene graph. As depicted in Figure 17.5, the concepts are progressively scored against all hierarchical layers, going from *higher* to *lower* concepts, *e.g.*, floors to regions to objects. In the case of regions, this procedure allows queries against a set of region representatives  $\{\mathbf{f}_{r,k}\}_{k=1}^K$  instead of queries against a single text embedding denoting the region category, which ultimately fosters robust region retrieval.

**Pose Maps.** While we adopted an object-centric perspective in this section, we note that a number of related methods instead store open-vocabulary semantic information at the level of camera poses or viewpoints [539, 1083, 1272]. This results in sparse spatial representations that can be leveraged for room segmentation and loop closure detections [539] and relocalization [1083]. Querying the map features returns the most relevant viewpoints and can be used to specify navigation goals.

The semantic and spatial overlaps between different viewpoints can localize specific object instances [1272], enabling object querying capabilities comparable to those of explicit object maps.

### 17.3.3 Implicit Functions

In Chapter 14 we saw the use of radiance fields for creating photo-realistic and geometric map representations using neural radiance field (NeRF) [761] and 3D Gaussian Splatting (3DGS) [551]. Numerous works have extended these representations to include open-set semantics. In general, most of the NeRF-based methods are trained to add language features to NeRF’s implicit radiance field map and methods using Gaussian Splatting assign embeddings to the Gaussians which can then be rasterized to the image plane. Since storing unique language embeddings to each Gaussian can lead to high memory usage (CLIP’s ViT-L/14 model for example uses language embeddings of size 768), it is common practice to either learn a compressed CLIP embedding or group Gaussians and assign a single CLIP vector to the group. We will look at LERF [554] and LangSplat [892] as examples of NeRF and 3DGS based approaches, respectively.

**LERF.** LERF adds an output to NeRF for a semantic embedding given a point  $\mathbf{x}$  and a scale  $s$ , which is the width of a cube centered at  $\mathbf{x}$ . View-dependency is omitted to enforce consistent output at different viewing directions. The embedding is rendered similarly to color in NeRF as covered in Chapter 14, with a frustum along each ray increasing proportionally from the initial scale  $s_{\text{img}}$  in the image to define the volume for rendering the semantic embedding, as shown in Figure 17.6. During training, to get CLIP embeddings for training supervision at multiple resolutions per image, LERF pre-computes CLIP embeddings at multiple randomly sampled scales of image crops (*i.e.*, multiple values for  $s_{\text{img}}$ ) for a given training image, resulting in an image pyramid of semantic embeddings. During training, rays are assigned ground-truth embeddings at randomly selected scales in the pyramid for supervision. To produce crisper semantics along boundaries of potential objects, LERF uses supervision from DINO [154] embeddings.

When querying a rendered image for similarity to a text embedding  $f_{\text{text}}$ , LERF computes a relevancy score to each language embedding  $f_{\text{render}}$  in the image as

$$\min_i \frac{\exp(\langle f_{\text{render}}, f_{\text{text}} \rangle)}{\exp(\langle f_{\text{render}}, f_{\text{canonical}}^i \rangle) + \exp(\langle f_{\text{render}}, f_{\text{text}} \rangle)}, \quad (17.7)$$

where  $f_{\text{canonical}}^i$  are embeddings from a set of canonical phrases such as “object”, “things”, “stuff”, and “texture.” Using the canonical phrases is intended to help denoise the relevancy score.

**LangSplat.** While LeRF computes CLIP vectors for multiple-sized image crops across an image pyramid to get semantic embeddings at multiple resolutions, Lang-

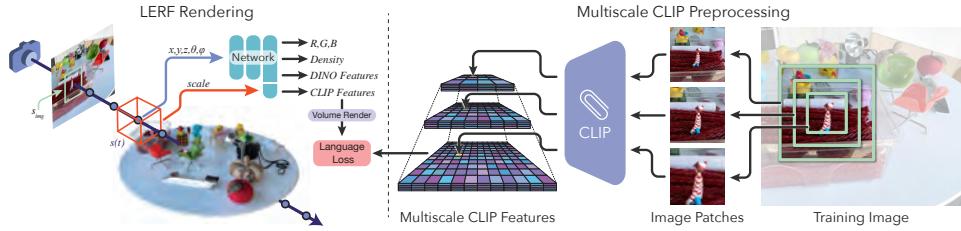


Figure 17.6 Reproduced from [554]. LERF [554] ( $\odot$ 2013 IEEE) trains a neural radiance field to render CLIP features. To get supervised semantic embeddings at multiple resolutions to be used during training, LERF precomputes a pyramid of embeddings by passing image crops at multiple scales to CLIP.

Splat [892] computes CLIP vectors of class-agnostic image segments obtained from Segment Anything Model (SAM) at three levels of granularity. Not only does this help capture local and global context, but it also induces crispness into the map of language embeddings, which makes it unnecessary to additionally use DINO supervision as in LERF.

Each 3D Gaussian is assigned an added attribute for each of the three levels of language embeddings. As assigning CLIP vectors to each Gaussian greatly increases memory requirements (LangSplat uses a CLIP model which has embeddings of size 512), LangSplat trains a scene-specific network that compresses CLIP vectors for a particular scene from their original size of 512 to size 3, and trains a decoder that recovers the original embedding. The intuition is that the semantic variation of a typical scene is much smaller than that of CLIP’s training corpus, and thus a smaller size embedding vector is sufficient. The 3D Gaussians then use compressed embedding vectors at the three levels of granularity as training supervision. Semantic rendering is performed as in the RGB case to splat the compressed embeddings at the three granularity levels to the image plane, and is then upscaled to the full-size embeddings. Querying is done using the same relevancy score as LERF, which in the case of LangSplat returns three scores (one for each granularity level) and the level with the highest score is selected when locating an object of interest.

#### 17.3.4 Task-Driven Representations

While open-set mapping can enable a broader range of concepts with richer descriptions than closed-set mapping, it brings up an interesting question of how to define objects. In the closed-set scenario, objects are defined by a list of labels such as “tool” and “cup”. In the open-set scenario, there is ambiguity of *what is the correct granularity to define objects*. For example, consider making an open-set semantic map of a scene consisting of a piano in an otherwise empty room. One possible map could have the piano as a single object. If the task of the agent using the

map is to move the piano, this would be an appropriate granularity. However, if the agent’s task is to play the piano, the scene must be mapped as a collection of more than 90 objects —considering each key and pedal as a separate object. An agent tasked with tuning the piano would need to consider the scene as hundreds of objects —considering the strings, tuning pins, and so forth. Likewise, if a forest should be represented as a single area of landscape or as branches, leaves, trunks, etc., remains ill-posed until we specify the tasks that the representation has to support. Most practitioners would agree that the correct scene representation depends on the tasks an agent must complete [729, 1016], but until the recent availability of vision-language foundation models, it has been unclear how to incorporate knowledge of tasks in practice.

We have previously looked at examples of selecting objects for an object-set map which typically assume objects can be defined by clustering regions based on semantic or geometric similarity (see for example, ConceptGraphs in Section 17.3.2). While this can approximate the correct object granularity for some scenes and applications, to more generally ensure the correct open-set map representation is generated, Clio [720] formally introduces the problem of *task-driven open-set mapping* as follows: given a set of tasks in natural language such as “clean backpacks” and “get condiment packets”, construct a map with objects at the correct granularity to support the tasks.

Clio (Figure 17.7) shows that the task-driven open-set mapping problem can be formalized mathematically using the classical Information Bottleneck theory [1095], where given a set of fine-grained primitive observations  $X$  (for example, in the extreme case this could be the set all pixels observed from every visual observation, or can be over-segmented class-agnostic masks), the goal is to form a compressed scene representation  $\tilde{X}$  (*i.e.*, the task-relevant objects) that preserves the necessary information needed to complete desired tasks  $Y$ .

Writing soft assignments in terms of probabilities  $p(\tilde{x}|x)$  (*i.e.*, the probability that a primitive  $x \in X$  is clustered into an object  $\tilde{x} \in \tilde{X}$ ), the task-driven clustering problem can be written using the Information Bottleneck as

$$\min_{p(\tilde{x}|x)} I(X; \tilde{X}) - \beta I(\tilde{X}; Y), \quad (17.8)$$

where  $I$  is the mutual information and  $\beta$  is a parameter that controls the trade-off between compression and preserving information about the tasks.

The Information Bottleneck requires defining  $p(y|x)$ , which in the case of task-driven mapping, describes the relevance between a primitive  $x$  and one of the given tasks  $y$ . To make the problem more tractable, Clio defines  $X$  as being a set of over-segmented 3D object primitives which are created by projecting class agnostic image segments,  $\text{Seg}(I)$  to 3D meshes and tracking them throughout frames. The primitives are assigned semantic embedding vectors by aggregating (through averaging) CLIP embedding vectors computed for each segment associated to the

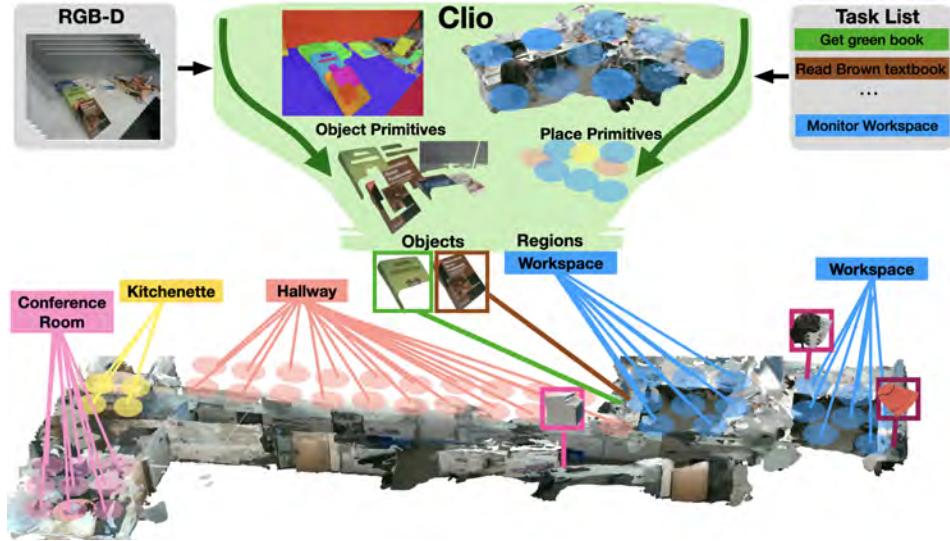


Figure 17.7 Reproduced from [720] (©2024 IEEE). Clio generates an open-set task-driven 3D scene graph given RGB-D images and a list of tasks in natural language. Clio clusters object primitives (derived from associating image segments in 3D) into task-relevant objects at the right granularity to support the task list. Likewise, the idea extends up the scene hierarchy by clustering place primitives (representing regions of free space) into task-relevant regions such as rooms.

primitive. Clio then uses the cosine score between image and text embeddings as a proxy for  $p(y|x)$ .

The optimization problem (17.8) can be solved incrementally using the Agglomerative Information Bottleneck algorithm [1014], which defines a graph where nodes correspond to primitives, and edges connect nearby primitives, and then gradually clusters primitives together. This produces a hard clustering assignment  $p(\tilde{x}|x)$ . More in detail, each edge  $(i, j)$  is assigned a weight,  $d_{ij}$  which measures the amount of distortion in information caused by merging primitive clusters  $\tilde{x}_i$  and  $\tilde{x}_j$  as:

$$d_{ij} = (p(\tilde{x}_i) + p(\tilde{x}_j)) \cdot D_{JS}[p(y|\tilde{x}_i), p(y|\tilde{x}_j)], \quad (17.9)$$

where  $p(\tilde{x}_i)$  and  $p(\tilde{x}_j)$  are computed by the algorithm (and intuitively store the number of primitives merged in clusters  $i$  and  $j$ ), and  $D_{JS}$  is the Jensen-Shannon divergence. At each iteration, edges are merged greedily based on their weight. As a result, Clio will merge nearby primitives into larger objects that are relevant to the tasks.

Clio extends the idea of task-driven mapping up the scene hierarchy by also clustering 3D regions of free space (referred to as places primitives) in the scene into task-relevant regions such as rooms, creating a hierarchical 3D scene graph with layers of object primitives, objects, place primitives, and regions.

While the mathematical representation in (17.8) is a general way to frame the task-driven mapping objective, in practice, solving it by approximating the information between image and text (*i.e.*, estimating  $p(y|x)$ ) and determining how to aggregate semantic knowledge across views are potential sources of error that can lead to incorrect representations. These two issues, which are currently active areas of research, are discussed in Bayesian Fields [719], which also provides a more grounded approach to estimate these probabilities. Bayesian Fields also uses a similar Agglomerative Information Bottleneck method as Clio, but instead of coarse meshes, uses a Gaussian Splatting map, where the task relevant objects,  $X$ , are groupings of 3D Gaussians which form high-resolution, photorealistic objects.

The current examples of task-driven mapping we have discussed require that the tasks be supplied in specific language such as “clean backpacks” (mostly due to limitations of CLIP), but looking ahead, task-driven mapping can be more broadly thought of as the case where robots determine on-the-fly how to represent and organize relevant information of a scene to complete their high-level mission objective —such as keeping a home clean or monitoring a factory. An initial work in this direction is Ashita [168], which simultaneously breaks down complex tasks into subtasks while grounding them into a 3D scene graph.

## 17.4 Further Readings & Recent Trends

In this section, we will try to project into the future, a difficult task given how rapidly things are evolving in this space.

### 17.4.1 Grounding Foundation Models with Maps

We saw in Section 17.2 how LLMs can be prompted to achieve a wide array of tasks conditioned on any textual information included in their prompts. Their rich prior knowledge of the world and reasoning abilities hold great promise for spatial AI, and there is significant interest in grounding LLMs in the physical world, potentially to deploy them as agents. This gives rise to a natural question that may guide research in this space in the coming years: *how can we fully integrate LLMs and map representations?* Here we highlight two other ways in which foundation models and map representations may become more integrated.

**Map Prompting.** Directly providing the map in text form to an LLM is sometimes possible. We previously saw an example of this approach with 3D scene graphs. When objects and edges are described with natural language by a VLM, a scene graph can be naturally represented as a structured text file (*e.g.*, JSON or YAML) and added to the prompt [407, 723, 1033]. Some sparse geometric descriptions of objects (*e.g.*, centroid position, size, bounding box coordinates) can, in principle, be included but how well LLMs can effectively leverage this information is unclear [723]. Such a modular system, where a VLM-based map is passed in text

form to an LLM, can be interpreted as a Socratic model [1], a modular framework in which pretrained models can be composed.

**Map API.** Textual map prompting fails to convey detailed geometric information about the world and is restricted to maps with a sparse, text-compatible structure. An alternative is to treat the map as a completely separate module and expose some functionalities through a symbolic map API. Examples of such functions could include queries such as `exists(object)`, `where_is(object)` or `distance(object_A, object_B)`, where inputs are described in natural language and the actual function implementations rely on the query mechanisms we discussed in Section 17.3. Map function calls can be interleaved with LLM calls and robot API calls to build complex and capable systems. As an example, NLMap [178] proposes a planning framework where an LLM breaks down user queries into different proposed objects and verifies object existence and location by querying an open-vocabulary map. The map API can also be leveraged for “tool calling”, a framework in which an LLM is free to call external functions and use their outputs during answer generation. LLMs can query open-vocabulary maps to generate robot plans [1272] or answer diverse scene queries expressed in natural language [512, 472].

#### 17.4.2 Revisiting the Question of the Need for Maps

With such impressive visual perceptual representations learned from internet-scale datasets, it is reasonable to revisit one of the first questions from Chapter I to this SLAM Handbook —the question of the need for SLAM maps, or at least explicit map representations. In this section, we will detail some ways in which explicit maps could, in some cases, be replaced directly by foundation models.

**Long-context VLMs.** Models such as Gemini and GPT4+ [1075, 5] can directly process sequences of RGB images that we would otherwise use to build a map, although to what extent long-context VLMs have a coherent understanding of the spatial structure underlying the images is unclear. OpenEQA [723] benchmarks various integrations of VLMs and scene graphs for embodied question answering tasks. They find that directly prompting GPT-4V with 50 frames outperforms using an explicit scene graph prompt on their benchmark. Mobility VLA [201] explores long-context VLMs in the context of navigation: it prompts a VLM with a full office tour (948 frames) and asks it to identify a high-level goal image given a user query. The goal image is then forwarded to a map-based navigation pipeline. The authors experimented with a map-free variant of their system, where the VLM was tasked to output navigation actions directly given the current observation and the office tour, and found this to be ineffective. Based on these results and others, we argue that, at present, *the need for an explicit map representation in the context of LLM or VLM-based physical agents appears to largely depend on the spatial and temporal horizons of the considered tasks* and remains an active area of research.

**Physics or Geometry-aware VLMs.** Some other approaches have attempted to enhance foundation models with an awareness of geometric and physical properties of the environment, something that has been demonstrated to be lacking out-of-the-box for most VLM models [45, 1165]. For example, [177] takes the perspective that the sensor is not *embodied* in the sense that it can be actuated or controlled and is only considering a single view. Nevertheless, these enhanced models have shown an impressive ability to reason about 3D space from a single camera image. However, while the performance on benchmarks such as OpenEQA is impressive, at present there is still an important gap between this and the level of embodied intelligence that we have seen demonstrated with the inclusion of SLAM in the pipeline. It is possible that this gap could be filled through other learning-based paradigms, such as imitation or reinforcement learning, a subject we will cover more thoroughly in the next section.

#### 17.4.3 A Foundation Model for Robotics?

Language and 2D vision foundation models have proved comparatively straightforward to build due to the availability of internet-scale data that can be used for training. Given the incredible advances that have ensued as a result of the representational power of these models, it is natural to consider if we can find a way to generate enough data to build foundation models for other domains. One great example is Dust3R [1161], discussed in Chapter 13, which has already had a significant impact on the field of 3D vision. But the question remains as to whether this is directly possible for robotics, and, if so, does this cause us to revisit the question of the need for explicit maps. Therefore, to close this chapter, we consider a recent effort in the robotics community to explicitly build a foundation model for robots.

*robotics foundation model*

The objective of such a *robotics foundation model* is to serve as a *base generalist policy* that can perform a diversity of skills across a variety of robot embodiments. The policy would have to learn how to somehow *implicitly* represent the world and then leverage this representation to execute tasks. The data therefore should be of the form of input-output pairs (sensor data and associated actions). As noted in Section 17.1, creating foundation models requires a massive amount of data, and this is preclusively difficult on real robot hardware. Nevertheless, there have been some efforts to collect this scale of data on both real hardware and using simulation.

In the next sections, we will briefly overview some of the recent efforts at collecting such large-scale robotics datasets, and then we will subsequently overview some of the models that have been built to consume these datasets with the objective of producing generally capable robot agents that do not require explicit world representations.

**Datasets and Benchmarks.** RT1: Robotics Transformer [118] was an early dataset effort comprising 130k robot trajectories of a fleet of 13 robots performing 700 different tasks over a period of 17 months. The “BridgeData V2” [1139] further

enhanced the diversity of tasks compared to RT1, including over 50k demonstrations of 13 skills over 24 environments. Although the hardware is fixed, there is an effort to randomize the external camera locations to provide robustness. The DROID dataset [558] was a multi-institutional effort to create a diverse dataset of robot interaction data that could be used primarily to train manipulation policies. All of these datasets and several others were integrated to create the Open-X-Embodiment dataset [226], which comprises over 1M episodes of 22 different robot embodiments performing 527 different “skills” in diverse environments and across 34 different university laboratories.

An important factor in scaling up the collection of data has been to build efficient and intuitive APIs for humans to provide demonstrations, often together with associated high-fidelity simulations. For example, DROID [558] leveraged a VR headset for teleoperation. The ALOHA [1074] setup is relatively low-cost and comprises two sets of synchronized manipulators so that humans can provide demonstrations that will be mimicked directly on the robot hardware. The “Universal Manipulation Interface” [200] is a low-cost handheld gripper setup that exactly matches a robot gripper and eye-in-hand sensor configuration and therefore can be easily used to collect robot demonstrations in a flexible and low-cost manner.

For the most part, the data in these aforementioned datasets takes the form of trajectories of sensor-action pairs  $\mathcal{D} = \{z_t, u_t\}_{t=1}^T$ , where  $T$  is the length of the trajectory. As such, they are really primarily targeting the learning of robot skills, such as grasping and manipulation, across a diversity of operating conditions and robot configurations. However, the objective is that complex tasks can be achieved either through increasingly complex demonstrations (combined with language conditioning), or by chaining together shorter demonstrations of sub-tasks.

**Model Categories.** There are several modeling paradigms that have been developed to *consume* the datasets presented in the previous section with the objective of producing *generalist* agents. These agents should be able to perform a wide array of abstractly specified tasks in a variety of environments and across a diverse set of robot configurations/morphologies.

A simple and intuitive approach to learn a behavior model from data is simply to perform behaviour cloning (BC): train a model in a supervised fashion to learn the mapping from observations to actions [878]. However, this approach is known to be brittle and prone to divergence since little sub-optimal data is contained in the demonstrations and small deviations in action from the demonstration lead to the model becoming out of distribution [942]. The “behavior transformer” model has shown initial promise in being able to overcome some of these challenges [990], even when trained and evaluated exclusively in simulation.

As a result, the majority of the models trained on these large-scale datasets, which form the “robotics foundation model” candidates, follow a similar structure as the foundation models introduced in Section 17.2, except now the output tokens

of the transformers are interpreted *directly as control actions*, referred to generally as “Vision Language Action” (VLA) models, as shown in Figure 17.8.

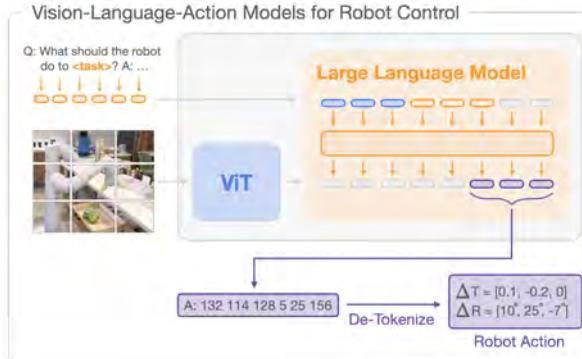


Figure 17.8 Figure reproduced from [119]. In a “Vision Language Action” model, a transformer is modified to directly output robot actions.

Perhaps the first such example was GATO [918], in which they showed that one generalist agent can potentially perform a huge diversity of tasks, including robot control, with a single model by tokenizing the input and output. Subsequently, PaLM-E [281], built an “embodied multimodal language model,” which was trained on vision-language data as well as robot manipulation data, to perform a diversity of language, motion planning, and manipulation tasks directly. RT1 [118] uses a FiLM image encoder rather than a ViT as was the case with PaLM-E, but otherwise the structure of the models is similar. RT2 [119] takes the data from PaLM-E and RT1 and combines them. RT1-X and RT2-X are variants of RT1 and RT2 trained on the Open-X-Embodiment dataset [226].

A variant of the standard VLA model is the “Action Chunking Transformer” (ACT) [1074]. Specifically designed to handle fine manipulation challenges with the ALOHA low-cost data collection platform, the ACT model outputs a sequence of actions rather than just an individual action (as was done in RT1), and has subsequently been shown to be an important modification to the original VLA transformer architecture to obtain better generalization performance [84].

Perhaps unsurprisingly, gains are potentially achievable by further integrating the representational power of existing VLMs and LLMs that we introduced in Section 17.2 into the language and vision encoders of these generalist policies. For example, the OpenVLA model is built on entirely open-source models, including VLM (DINOV2) and LLM (Llama 2) components, and, after finetuning for (15 days on a 64 A100 GPU cluster) produces a very performant model for the task of robot manipulation as compared to models trained on the Open-X-Embodiment dataset [572] alone. A similar approach was taken in the training of the  $\pi_0$  model [84].

A somewhat different paradigm for obtaining generalist robot policies is through the use of diffusion models [463]. A diffusion model is a type of generative model that learns to create data by reversing a gradual noising process. During training, it takes clean data and adds noise to it incrementally until it becomes pure noise. The model then learns how to reverse this process, denoising the noisy input in stages to recover the original data. When generating new samples, it starts from random noise and iteratively refines it to generate a sample from the training dataset. This process has been successfully applied to generate robot actions through a “diffusion policy” by representing a visuomotor policy as a conditional denoising process [199], and is currently a very active area of research.

It should also be noted that these two paradigms are not necessarily mutually exclusive. Octo [815], for example, leverages a large transformer for producing outputs which are fed to an “action head” that runs a forward diffusion process.

Reinforcement learning (RL) is also an increasingly popular paradigm for training robot policies. However, since the large datasets mentioned above do not contain an explicit notion of *rewards*, the reward would need to be inferred or learned as in an inverse RL setup. RL methods are particularly amenable to training in simulation where reward functions can be easily defined and evaluated, and where rollouts are relatively inexpensive to perform compared to real hardware. RL also has the potential to act as a fine-tuning mechanism that acts on the policies generated by the previously mentioned VLA or diffusion methods.

This section is by no means meant to be an exhaustive review of the state of the art of training generalist policies. This field is rapidly evolving with new, more capable models being released very frequently. As the capacity of these models grows, their ability to implicitly encode some representation of their environment also seems to improve, and may modify in what contexts we still need an explicit map representation.

#### 17.4.4 Concluding Remarks

It is undeniable that foundation models of all kinds have transformed robotics in a similar manner to many other fields. It remains to be seen, however, to what degree this progress continues. On the one hand, it seems conceivable that simply further scaling the robotics datasets and dedicated foundation models combined with internet-scale VLMs and LLMs could produce increasingly impressive capabilities that may render explicit map representations obsolete. However, it has been demonstrated that transformers in particular struggle to perform compositional or chain-of-thought reasoning [297]. For this reason, we are seeing the re-emergence of neural memory mechanisms, such as “structured state space models” [406], which are specifically designed to have a notion of statefulness, in much the same way as recurrent neural networks. But the limits of these neural memory architectures and how they become integrated with generalist policies are still unclear.

Equally as likely (or perhaps more so) at this time is that true generalization and scalability to compositional tasks in large and complex environments could be achieved through some form of explicit structure that is learned through a process such as SLAM. One view could be that, in fact, these two paradigms (explicit world modeling through SLAM and planning vs. generalist robotics policies) are entirely complementary. Perceiving, representing, and understanding the world is a prerequisite for taking action (at least in the types of complex problems in which we are interested), and this type of explicit world modeling should only help our ability to perform complex actions robustly.

# 18

## The Computational Structure of Spatial AI Systems

Andrew J. Davison

SLAM algorithms which map an environment and estimate a device’s position within it have enabled a range of important new products and capabilities which are already having an impact on the world, in areas such as industrial automation, home robotics, drones and virtual/augmented reality (VR/AR). However, we believe that this is just the start of something much bigger: the capability for artificial devices to build genuinely rich but efficient representations of their surroundings which enable them to interact with them in generally intelligent ways. We define this capability as *Spatial AI*, a term first coined in ‘FutureMapping’ [245].

*spatial AI*

In this chapter, we argue that Spatial AI capabilities are an ongoing evolution of the SLAM research that has been presented throughout this handbook. We predict and analyze the capabilities and computational structure of Spatial AI systems over the coming years, and in particular analyze the impact that machine learning and new sensing and computing hardware will have.

### 18.1 From SLAM to Spatial AI

#### 18.1.1 Intelligent Embodied Devices Need Spatial AI

The goal of a Spatial AI system is not abstract scene understanding, but continuously to build the right representations to enable real-time interpretation and general, intelligent action. The design of such a system will be framed at one end by task requirements on its performance, and at the other end by constraints imposed by the setting of the device in which it is to be used.

Let us consider the example of a mass market household robot product of the future, which is set the tasks of monitoring, cleaning, and tidying a set of rooms. The robot may be a humanoid, or other platform with general movement and manipulation capabilities. Its task requirements will include cleaning or sweeping surfaces and knowing when they are clean; recognizing, tidying, moving and manipulating objects; and dealing promptly and respectfully with humans by moving out of the way or assisting them. Meanwhile, its Spatial AI system, comprising sensors, proces-

sors and algorithms, will be constrained by factors including price, aesthetics, size, safety, and power usage, which must fit within the range of a consumer product.

As a second example, this time from the domain not of autonomous robotics but wearable assistive devices for humans (which with pleasing symmetry we may refer to as “IA”, Intelligence Augmentation), we imagine a future augmented reality system which can provide its wearer with a robust spatial memory of all of the places, objects and people they have encountered, enabling things such as easily finding lost objects, and the placing of virtual notes or other annotations on any world entity. This capability could enable the full context of a person’s life to be available automatically to an always-on Large Language Model assistant, and perhaps general amplification of that person’s intelligence capabilities. To achieve wide adoption, the device should have the size, weight, and form factor of a standard pair of spectacles (65g), and operate all day without needing a battery charge (<1W power usage).

There is currently a big gap between what such powerful Spatial AI systems need to do in useful applications, and what can be achieved with current technology under real-world constraints. There is much promising research on the algorithms and technology needed, but robust performance is still difficult even when expensive, bulky sensors and unlimited computing resources are available. The gap between reality and desired performance becomes much more significant still when the constraints on real products are taken into account. In particular, the size, cost, and power requirements of the computer processors currently enabling advanced robot vision are very far from fitting the constraints imposed by these applications we envisage.

### **18.1.2 Scene Representation and World Models**

We believe that the key to Spatial AI is *representation*. A representation is a set of values held in a computer’s memory which represent the state of an embodied device and the world around it in a way that is persistent, but adaptable, and always changing as the device acquires new information about the world, or the state of the world itself changes. Also, the device may choose to abstract or even “forget” parts of its representation for reasons of efficiency.

*world models*

The term ‘world model’ has recently become popular in machine learning research [423], and this term has very much the same meaning as scene representation. Many in machine learning consider world models a rather new domain of research, because most current trained neural network architectures are stateless, simply processing inputs and turning them deterministically into outputs. There is significant current research in robot learning on training such networks to perform tasks in an “end-to-end” manner, where inputs from sensors are turned directly into actions, in domains from autonomous driving to the manipulation of objects. These approaches are often extremely effective, but struggle to achieve long-horizon tasks.

World model research in neural networks aims to equip them with persistent memory modules which represent the changing state of a system and its environment, allowing simulation and longer-term planning.

Despite this recent new interest, the ideas of world models and scene representation of course have a long history in the more general world of AI, and SLAM research in robotics is the most clear example of where aiming to build persistent scene representations enables consistent, long-horizon behavior. Most obviously, SLAM maps enable reliable long-term navigation to multiple waypoints, or guaranteed area coverage, in applications such as autonomous drones or robot floor cleaners.

As SLAM expands beyond building minimal scene representations for localization towards dense, semantic and even object-level and physically predictive scene models, it will enable devices to simulate and plan in general ways. Model-predictive control (MPC) can take place in an incrementally built but accurate, up-to-date simulation of a robot's environment, allowing optimal and creative behavior, even for object manipulation.

Many machine learning researchers would assume that a 'world model' is something much more abstract than an explicit 3D map of a scene that is the typical output of a SLAM system, and we agree that there is much research to be done here on which world representations will enable true spatial intelligence while remaining efficient and practical. However, there are many advantages to representations which are rather explicit about the geometry and objects in a scene, in terms of efficiency, composability, interpretability, and generality.

We therefore make the following hypothesis: *When a device must operate for an extended period of time, carry out a wide variety of tasks (not all of which are necessarily known at design time), and communicate with other entities including humans, its Spatial AI system should build a general and persistent scene representation which is close to metric 3D geometry, at least locally, and is human understandable.* To be clear, this definition leaves a lot of space for many choices about scene representation, with both learned and designed elements, but rules out algorithms which make use of very specific task-focused representations, or completely abstract "black box" world models.

Our second hypothesis is that: *The usefulness of a Spatial AI system for a wide range of tasks is well represented by a relatively small number of performance measures.* That is to say that whether the system is to be used to guide the autonomous flight of a delivery drone in a tight space, or a household robot to tidy a room, or to enable an augmented reality display to add synthetic objects to a scene, then even though these applications certainly have different requirements and constraints, their Spatial AI systems will be largely similar, with differences specified by a small number of performance parameters. The obvious parameters describe aspects like global device localization accuracy and update latency, but we believe that there are other metrics which will be more meaningful for applications, like distance to

surface contact prediction accuracy, object identification accuracy, or tracking robustness. We will discuss performance metrics further in Section 18.8.

For the purposes of the rest of this chapter, we will therefore call such a module which incrementally builds and maintains a generally useful, close to metric scene representation, in real-time and from primarily visual input, and with quantifiable performance metrics, a *Spatial AI system*.

Judea Pearl, recently discussing efficient situated learning and the need to reason about causation [853], argued that “what humans possessed that other species lacked was a mental representation, a blue-print of their environment which they could manipulate at will to imagine alternative hypothetical environments for planning and learning”. We believe that Spatial AI is similarly what will set apart advanced robots and devices which can take embodied intelligence to the next level. Further, there is evidence that the spatial reasoning capabilities of humans are used more generally as a fundamental tool for intelligence, by organizing thinking in spatial ways, even for abstract concepts [71]; this may also be true for artificial systems.

#### **18.1.3 SLAM is Evolving into Spatial AI**

Research in SLAM, and especially visual SLAM, has long been driven by live demonstrations, with real-time visualizations, and the release of open source code. These live demos (of systems such as MonoSLAM, PTAM, KinectFusion, LSD-SLAM, SVO, ORB-SLAM, iMAP, Gaussian Splatting SLAM) have arguably been the most important markers of progress rather than the dataset results prominent in computer vision. This is because datasets often only capture limited aspects of a system’s performance (especially accuracy) while not highlighting robustness, efficiency, or flexibility, which are equally important and are readily assessed from a short real-time demo.

The level of scene representation that has been possible in real-time visual SLAM has gradually improved, from sparse features to dense maps and now increasingly semantic labels. Commercial SLAM provider Slamcore has referred to sparse localization, dense mapping, and semantic labeling as Levels 1, 2 and 3 capabilities respectively. Beyond this is perhaps the final destination of advances in scene representation, scene graphs with accurately segmented and physically simulated object instances making up geometry hierarchically (*cf.* Chapter 16 for further discussion). These are ongoing steps along SLAM’s evolution towards Spatial AI. Observing the steady and consistent progress of SLAM over more than 30 years, we have become confident that the operation of current and foreseeable SLAM systems is the best guide we have for the algorithmic structure of future Spatial AI.

We would like to emphasize that despite this level-based interpretation, the way that SLAM systems progress is usually not by the pure addition of layers to an already-working system. Rather, each change of representation causes the whole

system to be re-designed. Once a dense scene map is available, for instance, it can be used to implement accurate and robust direct tracking for camera localization; or semantic scene labeling can improve dense reconstruction quality. This kind of thinking is inherent to real-time systems research, where every part of a system affects every other part in a closed loop.

We therefore believe strongly that ongoing SLAM research is the best model we have for the development of Spatial AI.

## 18.2 Overall Computational Structure

As a sensor platform carrying at least one camera and other sensors moves through the world, its motion either under active control or provided by another agent, the essential way that a Spatial AI system of any variety works can be summarized as follows:

- 1 Our system will comprise outward looking sensors such as cameras, and supporting sensors such as an IMU, closely integrated with a processing architecture in a low-power package which is embedded in a mobile entity such as a robot or AR system.
- 2 In real-time, the system must maintain and update a world model, with geometric and semantic information, and estimate its position within that model, from primarily or only measurements from its on-board sensors.
- 3 The system should provide a wide variety of task-useful information about ‘what’ is ‘where’ in the scene. Ideally, it will provide a full semantic level model of the identities, positions, shapes and motion of all of the objects and other entities in the surroundings.
- 4 The representation of the world model will be close to metric, at least locally, to enable rapid reasoning about arbitrary predictions and measurements of interest to an AI or IA system.
- 5 It will probably retain a maximum quality representation of geometry and semantics only in a focused manner; most obviously for the part of the scene currently observed and relevant for near-future interaction. The rest of the model will be stored at a hierarchy of residual quality levels, which can be rapidly upgraded when revisited.
- 6 The system will be generally alert, in the sense that every incoming piece of visual data is checked against a forward predictive scene model: for tracking, and for detecting changes in the environment and independent motion. The system will be able to respond to changes in its environment.

The key computational quality of this approach is its *closed loop* nature, where the world model is persistent and incrementally updated, representing in an abstracted form all of the useful data which has been acquired to date, and is used in the real-time loop for data association and tracking. This is in contrast with vision systems

which perform incremental estimation (such as pure visual odometry, where camera motion is estimated from frame to frame but long term data structures are not retained), or which can only achieve global consistency with off-line, after-the-fact, batch computation. That is not to say that in a closed loop Spatial AI system every computation should happen at a fixed rate, but more importantly that it should be available when needed to allow real-time operation of the whole system to continue without pauses.

The next element of our high-level thinking is to identify the core ways that we can achieve all of this with high performance but low power requirements. We believe that the key to efficient processing in Spatial AI is to identify the graphs of computation and data movement in the algorithms required, and as far as possible to make use of or design processing hardware which has the same properties, with the particular goal of minimizing data movement around the architecture. We need to identify the following things: What is stored where? What is processed where? What is transmitted where, and when?

We will not attempt here to draw strong parallels with neuroscience, but clearly there is much scope for relating the ideas and designs we discuss here in artificial Spatial AI systems with the vision and spatial reasoning capabilities and structures of biological brains. The human brain apparently achieves high performance, fully ‘embedded’ semantic and geometric vision using less than 10 Watts of power, and certainly its structures have properties which mirror some of the concepts we discuss. We will leave it to other authors to analyze the relationships further; this is mostly due to our lack of expertise in neuroscience, but also partly to a belief that while artificial vision systems clearly still have a great deal to learn from biology, they need not be designed to replicate the performance or structure of brains. We consider the Spatial AI computation problem purely from the engineering point of view, with the goal of achieving the performance we need for applications while minimizing resources. It should surely not be surprising that some aspects of our solutions should mimic those discovered by biological evolution, while in other respects we might find quite different methods due to two contrasts: firstly between the incremental ‘has to work all of the time’ design route of evolution and the increased freedom we have in AI design; and secondly between the wetware and hardware available as a computational substrate.

Before making this computational analysis more concrete, in terms of proposing a generic design for Spatial AI implementations, we need to consider two important topics: (i) state estimation vs. machine learning, and (ii) the landscape of future processor and sensor hardware. We will discuss these in the next two sections.

### **18.3 State Estimation and Machine Learning in Spatial AI**

Much of this handbook covers approaches to SLAM which are based on human-designed algorithms and representations; predominantly ‘state estimation’ methods

based on probability theory. In the later chapters, we see the rise of machine learning techniques of many types. The key machinery of machine learning in SLAM is the artificial neural network. SLAM is notable within the broader picture of artificial intelligence in that it has not yet been completely dominated by ML. The best performing systems for localization and sparse mapping are still mostly hand-designed, and dense and semantic SLAM systems use a balance, where ML is used to replace part of or add to the functionality of state estimation. However, the gradual rise of ML techniques to cover more of the functionality of Spatial AI has been undeniable.

In some of the first uses of ML in SLAM, a learned module was used to add a layer of functionality but did not interact directly with the state estimation components. An example is SemanticFusion [744], which used CNN-based semantic image segmentation to add labels to a the dense geometric map produced by ElasticFusion [1181]. Soon though, it became clear that learned components could play important roles in the core geometric estimation loop of SLAM, and this has taken several key forms, including:

- 1 Networks trained to predict geometric properties from single images, such as depth maps, normal maps, or object segmentation.
- 2 Components for updating standard geometric estimates, like poses and point clouds, as used in DROID-SLAM; these networks often perform operations such as matching or geometric optimization.
- 3 The latest networks which produce higher-quality geometric predictions from multi-image input, such as DUSt3R and VGGT.

Although neural networks can directly predict geometry, such as depth maps from a single image, it has proven difficult to use these outputs in SLAM because single-view predictions often contain large systematic errors, and have poorly understood uncertainty. A promising branch of this approach is to train networks to predict somewhat weaker properties than raw depth, such as coded depth CodeSLAM [91] or depth covariance [269], allowing low-dimensional multi-view optimization to perform consistent dense geometry fusion.

Perhaps the end goal of this single-view research is to develop a network which can turn an input image into a reliable set of semantic, object-like entities which can be then used to populate an efficient, abstract 3D object map or scene graph which is subject to ongoing, dynamic multi-view optimization and updating. This approach was pioneered in SLAM by SLAM++ [961], which built an efficient scene graph map directly at the level of recognized 3D objects, though with the limitation that a pre-defined 3D model of every object was needed. A modern generalization of this approach is SuperPrimitives [741] which uses generic single-image object segmentation and local reconstruction to enable a long-term map of arbitrary 3D object chunks.

The appeal of machine learning in Spatial AI has perhaps two key aspects. First,

neural networks can be trained to carry out tasks which have proven very difficult to design by hand, such as scene labeling or depth prediction. Second, they compress iterative computation into an efficient, computationally simple form, which can run feed-forward and often fast on modern processors.

We should naturally ask the question of whether the whole of SLAM and Spatial AI will ultimately be performed end-to-end by a neural network, with no hand-designed representations or state estimation at all. Of course many researchers believe in this idea, and there are increasingly impressive pieces of work from the early demonstrations of end-to-end visual odometry DeepVO [1160, 1296] to networks such as VGGT which outputs 3D point clouds directly from a set of unposed images.

This work will surely continue to improve, but we believe that hybrid methods combining learning and state estimation will continue to be preferable for the foreseeable future, for several reasons. First, the issue of understanding the uncertainty of network predictions remains, and we will need to keep the ability to update and fuse data into models. If we have a network which can produce a 3D scene model from 100 images, how should we update it when one more image is available? Surely we should not need to run the whole network again with 101 images. As soon as we accept that long-term representation and fusion is needed, then we need the tools of probabilistic state estimation, and there is a strong motivation towards modular scene representations.

Ultimately, the Spatial AI systems of the mid-term future will surely have learned and designed components so tightly integrated that it is hard to tell which is which; and this position may be reached either by adding more and more learned modules to a designed system, or by adding structure to a purely learned system. What still remains important in either case, and at the center of our interest in this chapter, is the storage and computational structure of the whole system.

## **18.4 The Future Landscape of Processor and Sensor Hardware**

### ***18.4.1 Processors***

*processor architecture*

SLAM research was for many years conducted in the era when single core CPU processors could reliably be counted on to double in clock speed, and therefore serial processing capability, every 1–2 years. In recent years this has stopped being true. The strict definition of Moore’s Law, describing the rate of doubling of transistor density in integrated circuits, has continued to hold into the current era. What has changed is that this can no longer be proportionally translated into serial CPU performance, due to the breakdown of another less well known rule of thumb called Dennard Scaling, which states that as transistors get smaller their power density stays constant. When transistors are reduced down to today’s nanometre sizes, not so far from the size of the atoms they are made from, they leak current and heat

up. This ‘power wall’ limits the clock speed at which they can reasonably be run without overheating uncontrollably to something around 4GHz.

Processor designers must therefore increasingly look towards alternative means than simply faster clocks to improve computation performance. The processor landscape is becoming much more complex, parallel and specialized, as described well in Sutter’s online article ‘Welcome to the Jungle’ [1059]. Processor design is becoming more varied and complex even in ‘cloud’ data centers. Pressure to move away from CPUs is even stronger in embedded applications like Spatial AI, because here power usage is a critical issue, and parallel, heterogeneous, specialized processors seem to be the only route to achieving the computational performance Spatial AI needs within power restrictions which will fit real products. So while current embedded vision systems, *e.g.*, for drones, often use CPU-only implementations of SLAM algorithms (rather than requiring GPUs), we believe that this is not the right approach for the longer term. While current desktop GPUs are certainly power-hungry, Spatial AI must fully embrace parallelism. However, GPUs are only the beginning of the wide space of processor designs that will emerge over the coming years. We highly recommend the recent PhD thesis of Julien Martel for ambitious thinking about this whole area [731].

Mainstream geometric computer vision started to take advantage of parallel processing in the form of GPUs nearly 20 years ago (*e.g.*, [877]), and in Spatial AI this led to breakthroughs in dense SLAM [797, 798]. The SIMD (Single Instruction, Multiple Threads) parallelism that GPUs provide is well suited to elements of real-time vision where the same operation needs to be applied to every element of a regular array in image or map space. Concurrently, GPUs were central to the emergence of deep learning in computer vision [606], by providing the computational resource to enable neural networks of sufficient scale to be trained to finally prove their worth in significant tasks such as image classification.

The move from CPUs to GPUs as the main processing workhorse for computer vision is only the beginning of how processing technology is going to evolve. We foresee a future where an embedded Spatial AI system will have a heterogeneous, multi-element, specialized architecture, where low power operation must be achieved together with high performance. A standard SoC (system-on-chip) for embedded vision ten years from now, which might be used in a personal mobile device, consumer robot or AR headset, will be likely to still have elements which are similar in design to today’s CPUs and GPUs, due to their flexibility and the huge amount of useful software they can run. However, it is also likely to have a number of specialized processors optimized for low power real-time vision.

The key to efficient processing which is both fast and consumes little power is *to divide computation between a large number of relatively low clock-rate or otherwise simple cores, and to minimize the movement of data between them*. A CPU pulls and pushes small pieces of data one by one from and to a separate main memory store as it performs computation, with local caching of regularly used data the only

mechanism for reducing the flow. Programming for a single CPU is straightforward, because any type of algorithm can be broken down into sequential steps with access to a single central memory store, but the piece by piece flow of data to and from central memory has a huge power cost.

More efficient processor designs aim to keep processing and the data operated on close together, and to limit the transmission of intermediate results. The ideal way to achieve this is a close match between the design of a processor/storage architecture and the algorithm it must run. A GPU certainly has large advantages over a CPU for many computer vision processing tasks, but in the end a GPU is a processor designed originally for computer graphics rather than vision and AI. Its SIMD architecture can efficiently run algorithms where the same operation is carried out simultaneously on many different data elements. In a full Spatial AI system, there are still many aspects which do not fit well with this, and a joint CPU/GPU architecture is currently needed with substantial data transfer between the two.

While it is relatively accessible to design custom ‘accelerator’ processors which could implement certain specific low-level algorithms with high efficiency, there has been relatively little work until recently on thinking about the whole computational structure of closed-loop embedded systems like Spatial AI. It is certainly true now that low power vision is seen as an increasingly important aim in industry, and custom processors to achieve this have been developed such as Movidius’ Myriad series. These processors combine low power CPU-like, DSP-like and custom elements in a complete package. The ‘HPU’ custom-designed by Microsoft for their original Hololens AR headset is rather similar in design. More recently, the Apple Vision Pro uses a custom Apple R1 co-processor for real-time sensor input processing. Meta’s ARIA Gen 2 smart glasses research device has custom silicon for ‘ultra low power and on-device machine perception’, so that SLAM, eye tracking, hand-tracking and speech recognition can all operate on-device without the need for an external battery. The details of these commercial chips and the algorithms that run on them are confidential, but we can imagine that they are custom designed for efficient operation of small neural networks, other image processing and some level of general probabilistic optimization.

If we try to look further ahead, we can conceive of processor designs which offer the possibility of a much closer match between architecture and algorithms. Highly relevant to our aims are major efforts which are now taking place on new ways of doing large-scale processing by being made up from large numbers of independent and relatively low-spec cores with the emphasis on communication. SpiNNaker [348] is a major research project from the University of Manchester which aims to build machines to emulate biological brains. It has produced a prototype machine made up from boards which each have hundreds of ARM cores, and with up to a million cores in total. With the rather different commercial aim of providing an important new type of processor for AI, Graphcore is a UK company developing ‘Intelligence

Processing Unit' (IPU) processors which comprise thousands cores on a single chip, each of which with which is able to run an independent program on its own local memory, with an efficient programmable interconnect structure. Other companies such as Tenstorrent and Cerebras have interesting novel chip designs.

The focus of these projects has largely been efficient implementation of neural networks, in the case of both SpiNNaker and Graphcore with the belief that the important matter is the overall topology of a large number of cores, each performing different operations but highly and efficiently inter-connected in a graph configuration adapted to the use case. These designs have not taken strong decisions about the type of processing carried out at each core, or the type of messages they can exchange, with the desire to leave these matters to the choice of future programmers. This is as opposed to more some explicitly neuromorphic architectures aiming to implement particular models of the operation of biological brains, such as IBM's Truenorth project.

Such architectures are not yet close to ready for embedded vision, but seem to offer great long term potential for the design of Spatial AI systems where the graph structure of the algorithms and memory stores we use can be matched to the implementation on the processor in a custom and potentially highly efficient way. We will consider this in more detail later on.

But we also believe that we should go further than thinking of mapping Spatial AI to a single processor, even when it has an internal graph architecture. A more general concept of a graph applies to communication to cameras and other sensors, actuators and other outputs; to other independent robots and devices sharing the same environment; and potentially entirely off-board computing resources in the cloud.

#### 18.4.2 Sensors

Cameras are the most important sensors in Spatial AI, and the concept of a camera is today becoming increasingly broad with ongoing innovation by sensor designers; for our purposes we consider any device which essentially captures an array of light measurements to be a visual sensor. Most are passive in that they record and measure the ambient light which reaches them from their surroundings, while another large class of cameras emit their own light in a more or less controlled fashion. In Spatial AI, many types of camera have been used, with the most common being passive monocular and stereo camera rigs, and depth cameras based on structured light or time of flight concepts. Every camera design represents a choice in terms of the quality of information it provides (measured in such ways as spatial and temporal resolution and dynamic range), and the constraints it places on the system it is used in such as size and power usage. In previous work [428] we studied some of the trade-offs possible between performance and computational cost in the Spatial AI sub-problem of real-time tracking.

A particularly promising sensor for Spatial AI, as already discussed in this handbook, is the event camera , which removes redundancy by sensing and transmitting only changes in intensity. An event stream encodes the information content of video at a much lower bit-rate, while offering advantages in temporal and intensity sensitivity. This is surely only the starting point for coming rapid changes in image sensor technology, where low power computer vision will be an increasingly important driver.

One significant ongoing academic project in this space is the SCAMP series of vision chips with in-plane processing from the University of Manchester (see [732] for an introduction). The SCAMP5 chip runs at 1.2W and has an image resolution of  $256 \times 256$ , with each pixel controlled by and processable by per-pixel processing. Using analog current-mode circuits, summation, subtraction, division, squaring, and communication of values with neighboring pixels can be achieved extremely rapidly and efficiently to permit a significant level of real-time vision processing completely on-chip. Ultra-low power operation can alternatively already be achieved in applications where low update rates are sufficient.

As we will discuss later on, the range of vision processing which could eventually be performed by such an image plane processor is still be to fully discovered. The obvious use is in front-end pre-processing such as feature detection, local motion tracking, or segmentation. We believe that the longer term potential is that while a central processor will be required for full model-based Spatial AI, close-to-sensor processing can interact fully with this via two-way low communication, with the main aim of reducing the bit-rate needed between the sensor and main processor and therefore the communication power requirements.

Finally, when considering the evolution of the computing resources for Spatial AI, we should never forget that cloud computing resources will continue to expand in capacity and reduce in cost. All future Spatial AI systems will likely be cloud-connected most of the time, and from their point of view the processing and memory resources of the cloud can be assumed to be close to infinite and free. What is not free is communication between an embedded device and the cloud, which can be expensive in power terms, particularly if high bandwidth data such as video is transmitted. The other important consideration is the time delay, typically of significant fractions of a second, in sending data to the cloud for processing. There may also be applications where an always-on, high-bandwidth cloud connection is unfeasible (*e.g.*, space, undersea, underground).

### **18.5 Mapping Spatial AI Graphs to Hardware**

We now turn more specifically to a design for the architecture of a Spatial AI device. Despite the clear potential for cloud-connected shared mapping, here we choose to focus purely on a single device which needs to operate in a space with only on-board

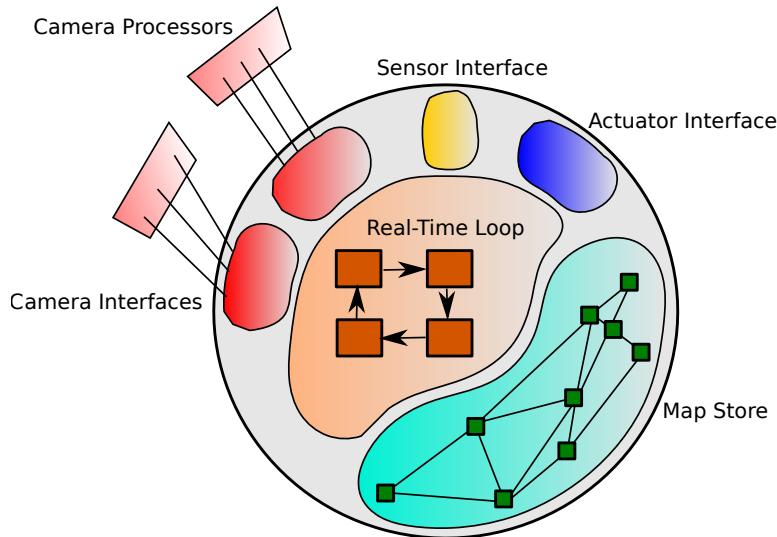


Figure 18.1 Spatial AI brain: how the representation and processing graph structures of a general Spatial AI system might map to a graph processor. The key elements we identify are the real-time processing loop, the graph-based map store, and blocks which interface with sensors and output actuators. Note that we envision additional ‘close to the sensor’ processing built into visual sensors, aiming to reduce the data bandwidth (eventually in two directions) between the main processor and cameras, which will generally be located some distance away.

resources, because this is the most generally capable setup which could be useful in any application and not require additional infrastructure.

We argue that in the overall architecture of our Spatial AI computation system, the long-term key to efficient performance is to match up the natural graph structures of our algorithms to the configuration of physical hardware. As we have seen, this reasoning leads to the use of ‘close to the sensor’ processing such as in-plane image processing, and the attempt to minimize data transfer from sensors and towards actuators or other outputs using principles such as events.

However, we still believe that the bulk of computation in an embedded Spatial AI system is best carried out by a relatively centralized processing resource. The key reason for this is the essential and ever-present role of an incrementally built and used world model representing the system’s knowledge of its state and that of its environment. Every new piece of data from possibly multiple cameras and sensors is ultimately compared with this model, and either used to update it or discarded if the data is not relevant to device’s short or long-term goals.

### **18.5.1 World Model Processing**

What we are anticipating for this central processing resource is however far from the model of a CPU and RAM-like memory store. A CPU can access any contents of its RAM with a similar cost, but in Spatial AI there is much more structure present, both in the locality of the data representing knowledge of the world and in the organization of the computation workload involved in incorporating new data.

The graph structures of processing and storage should be built into the design of the central world model computation unit, which should combine storage and processing in a fully integrated way. There are already significant efforts on new ways to design architectures which are explicitly and flexibly graphlike.

To focus on one processor concept, Graphcore's IPU or graph processor is designed to efficiently carry out AI workloads which are well modeled as operations on sparse graphs, and in particular when all of the required storage is itself also held *within* the same graph rather than in an external memory store. An IPU chip has a large number of independent cores (in the thousands), each of which can run its own arbitrary program and has its own local memory store, and then a powerful communication substrate such that the cores can efficiently send messages between themselves. When a program is to be run on the IPU, it is first compiled into a suitable form by analysis of the patterns of computation and communication it requires. A suitable graph topology of optimized locations of operations, data, and channels of communication is generated.

In Figure 18.1 we have made a first attempt at drawing a ‘Spatial AI brain’ model which is analogous to the way Graphcore compiles a computation graph onto the IPU. The disc contains the modules we anticipate running inside the main processor. One of the two main areas is the map store, which is where the current world model is stored. This has an internal graph structure relating to the geometry of the world. It will also contain significant internal processing capability to operate locally on the data in the model, and we will discuss the role of this shortly. The second main area is the real-time loop, which is where the main real-time computation connecting the input image stream to the world model is carried out. This has a processing graph structure and must support large real-time data flows and parallel computation on image/map structures so is designed to optimise this.

The main processor also has additional modules. There are camera interfaces, the job of each of which is to model and predict the data arriving at the sensor to which it is connected. This will then be connected to the camera itself, which the physical design of a robot or other device may force to be relatively far from the main processor. The connection may be serial or along multiple parallel lines.

We then imagine that each camera will have its own ‘close-to-sensor’ processing capability built in, separated from the main processor by a data link. The goal of modelling the input within the main processor is to minimize actual data transfer to the close-to-sensor processors. It could be that the close-to-sensor processor

performs purely image-driven computation, in a manner similar to the SCAMP project, and delivers an abstracted representation to the main processor. Or, there could be bi-directional data transmission between the camera and main processor. By sending model predictions to the close-to-sensor processors, they know what is already available in the main processor and should report only differences. This is a generalization of the event camera concept. An event camera reports only changes in intensity, whereas a future optimally efficient camera should report places where the received data is *different from what was predicted*. We will discuss close-to-sensor processing further in Section 18.5.3.

#### 18.5.1.1 World Model

The graph structure of world models has been recognized and made use of by many important SLAM methods (*e.g.*, [533, 594, 961, 307, 751]).

As a robot moves through and observes the world, a SLAM algorithm detects, tracks, and inserts into its map features which are extracted from the image data. Note that we use the word ‘feature’ here in a general sense to mean some abstraction of a scene entity, and that we are not confining our thinking to sparse point-like landmarks. As abstraction in Spatial AI increases, these features are likely to be objects or other semantic entities. Each feature in the scene has a region of camera positions from which it is measurable. A feature will not be measurable if it is outside of the sensor’s field of view; if it is occluded; or for other reasons such that its distance from the camera or angle of observation are very different from when it was first observed.

This means that as the robot moves through a scene, features become observable in variable overlapping patterns. As measurements are made of the currently visible features, estimates of their locations are improved, and the measurements are also used to estimate the camera’s motion. The estimates of the locations of features which are observed at the same time become strongly correlated with each other via the uncertain camera state. Features which are ‘nearby’ in terms of the amount of camera motion between observing them are still correlated but somewhat less strongly; and features which are ‘distant’ in that a lot of motion (and SLAM based on intermediate features) happens between their observation are only weakly correlated.

The probabilistic joint density over feature locations which is the output of SLAM algorithms can be efficiently represented by a graph where ‘nearby’ features are joined by strong edges, and ‘distant’ ones by weak edges. A threshold can be chosen on the accuracy of probabilistic representation which leads to the cutting of weaker edges, and therefore a sparse graph where only ‘nearby’ features are joined.

One way to do SLAM is not to explicitly estimate the state of scene features, but instead to construct a map of a subset of the historical poses that the moving camera has been in, and to keep the scene map implicit. This is usually called pose-graph SLAM (*cf.* Chapter I), and within this kind of map the graph structure

is obvious because we join together poses between which we have been able to get sensor correspondence. Poses which are consecutive in time are joined; and poses where we are able to detect a revisit after a longer period of time are also joined (*i.e.*, loop closures). Whether the graph is of historic poses, or of scene features, its structure is very similar, in that it connects either ‘nearby’ poses or the features measured from those poses, and there is not a fundamental difference between the two approaches.

This leads us to conclude that *the most likely representation for Spatial AI is to represent 3D space by a graph of features, which are linked in multi-scale patterns relating to camera motion and together are able of generating dense scene predictions*. Within the main processor, a major area will be devoted to storing this map, in a manner which is distributed around potentially a large number of individual cores which are connected in a topology to mirror the map graph topology. In SLAM, the map is defined and grown dynamically, so the graph within the processor must either be able to change dynamically as well, or must be initially defined with a large unused capacity which is filled as SLAM progresses.

Importantly, a significant portion of the processing associated with large-scale SLAM can be built directly into this graph. This is mainly the sort of ‘maintenance’ processing via which the map optimizes, refines, and abstracts itself; including:

- 1 Feature clustering; object segmentation and identification.
- 2 Loop closure detection.
- 3 Loop closure optimization.
- 4 Map regularization (smoothing).
- 5 Unsupervised clustering to discover new semantic categories.

With time, data, and processing, a map which starts off as dense geometry and low-level features can be refined towards an efficient object-level map. These operations will run with very high parallelism and data locality. Smoothing and segmentation for instance take place at each part of the graph separately, while global map optimization can be implemented via message passing, as we will explain in Section 18.6. Most importantly, all of these processes can take place in a manner which is internal to the map store itself.

The on-chip memory of current graph processors like Graphcore’s IPU is fully distributed among the processing tiles, and the total capacity is large (on the order of around 1MB per tile, or around 1GB total over 1000 tiles), but not huge (certainly when compared with standard off-processor RAM), and therefore there should be an emphasis on rapidly abstracting the map store towards an efficient long-term form. Future processors may however have much more distributed storage.

### 18.5.2 Real-Time Loop

We make a hypothesis that the core computation graph for the tightest real-time loop of future SLAM systems will have many elements which are familiar from today's systems. Specifically the dense SLAM paradigm introduced by Newcombe *et al.* [796, 798, 797] is at the center of this, because this approach aims to model the world in dense, generative detail such that every new pixel of data from a camera can be compared against a model-based prediction. This allows systems which are generally ‘aware’, since as they continually model the state of the world in front of the camera, they can detect when something is out of place with respect to this model, and therefore dense SLAM systems are now being developed which are moving beyond static scenes to reconstruct and track dynamic scenes [799, 954]. Dense SLAM systems can also make the best possible use of scene priors, which will increasingly come from learning rather than being hand-designed.

Some of the main computational elements in the real-time loop are:

- 1 Empirical labeling of images to features, usually via a neural network.
- 2 Rendering: producing a dense prediction from the world map into image space.
- 3 Tracking: aligning a prediction with new image data, including finding outliers and detecting independent movement.
- 4 Fusion: fusing updated geometry and labels back into the map.
- 5 Self-supervised learning from the running system.

Architecturally, this processing depends on both the live input from cameras and other sensors, and the world model, and must therefore in some sense take place ‘between’ them. Data from the map store is needed for *rendering*, when a predicted view of the scene is needed for tracking against new image data, and for *fusion*, when information (geometry and labels) acquired from the new data is used to update the map contents. This is generally massively parallel processing which is familiar from GPU-accelerated dense SLAM systems, and these functions can be defined as fixed elements in a computational graph which use a number of nodes and access a significant fraction of the main computational resource. Functions such as segmentation and labeling of input images could also be implemented here, though as we will discuss shortly these would be more sensibly located outside of the main processor by close-to-sensor processing.

The most difficult issue in applying graph processing to the real-time loop is the fact that correspondence with the relevant part of the graph-based map store for these operations changes continuously due to motion. This seems to preclude defining an efficient, fixed data path to the distributed memory where map information will be stored. Although there will be internal processing happening in the map store, this will be focused on maintenance and it does not seem appropriate to move data rapidly around the map store, for instance such that the currently observed part of the map is always available in a particular graph location.

Instead, a possible solution is to define special interface nodes which sit between the real-time loop block and the map store. These are nodes focused on communication, which are connected to the relevant components of real-time loop processing and then also to various sites in the map graph, and may have some analogue in the hippocampus of mammal brains. If the map store is organized such that it has a ‘small world’ topology, meaning that any part is joined to any other part by a small number of edge hops, then the interface nodes should be able to access (copy) any relevant map data in a small number of operations and serve them up to the real-time loop.

Each node in the map store will also have to play some part in this communication procedure, where it will sometimes be used as part of the route for copying map data backwards and forwards.

#### **18.5.3 Processing Close to the Image Plane**

*on-sensor vision*

A robot or other device will have one or more cameras which interface with the main processor, and we believe that the technology will develop to allow a significant amount of processing to occur either within the sensors themselves or nearby, with the key aim of reducing the amount of redundant data which flows from the cameras.

The pixels which make up the image sensor of a camera have a regular, usually rectangular geometry. While each of these pixels is normally independently sensitive to light intensity, many vision algorithms make use of the fact that the output of nearby pixels tends to be strongly correlated. This is because nearby pixels often observe parts of the same scene objects and structures. Most commonly, a regular graph in which each pixel is connected to its four (up, down, left, right) neighbors is used as the basis for smoothing operations in many early vision problems such as dense matching or optical flow estimation (e.g. [876]). The regular graph structure of images is also taken advantage of by the early convolutional layers of CNNs for all kinds of computer vision tasks. Here the multiple levels of convolutions also acknowledge the typical hierarchical nature of local regularity in image data.

Most straightforwardly, a sensor with in-plane processing similar to SCAMP5 [732] could be used to carry out purely bottom-up processing of the input image stream; abstracting, simplifying, and detecting features to reduce it to a more compact, data-rich form. Calculations such as local tracking (e.g., optical flow estimation), segmentation and simple labeling could also be performed. This has biological analogies with the ‘early vision’ processing carried out by the retina and optic nerve.

Tracking using in-plane processing is an interesting problem. In-plane processing is good for problems where data access can be kept very local, so estimating local image motion (optical flow), where the output is a motion vector at every pixel, is well suited. At each update, the amount of image change locally can be augmented using local regularization where smoothness is applied based on the differences of neighboring pixels. If we look at the parallel implementation of an algorithm like

Pock’s TV-L1 optical flow [876], we see that it involves pixelwise-parallel operations, where purely parallel steps relating to the data term alternate with regularization steps involving gradient computation, which could be achieved using message passing between adjacent neighbors. So such an algorithm is an excellent candidate for implementation on an in-plane processor.

More challenging is the tracking usually required in SLAM, where from local image changes we wish to estimate consistent global motion parameters relating to a model. This could be instantaneous camera motion estimation, where we wish to estimate the amount of global rotation for instance between one frame and the next via whole image alignment [704], or tracking against a persistent scene model as in dense SLAM [695, 796]. When dense tracking is implemented on standard processors, it involves alternation of purely parallel steps for error term computation across all pixels with reduction steps where all errors are summed and the global motion parameters are updated. The reduction step, where a global model is imposed, plays the role of regularization, but the big difference is that the regularization here is global rather than local.

In a modern system, such global tracking is usually best achieved by a combination of GPU and CPU, and therefore a regular (and expensive) transfer of data between the two. But we do not have this option if we wish to use in-plane processing and keep all computations and memory local. Our main option for not giving up on data locality is to give up on guaranteed global consistency of our tracking solution, but to aim to converge towards this via local message passing. Each pixel could keep its own estimate of the global motion parameters of interest, and after each iteration share these estimates with local neighbors. We would expect that global convergence would eventually be reached, but that after a certain number of iterations that the values held by each pixel would be close enough that any single pixel could be queried for a usable estimate. Convergence would be much faster if the in-plane processor also featured some longer data-passing links between pixels, or more generally had a ‘small world’ pattern of interconnections.

Turning to the questions of labeling using local processing, this is certainly feasible but a problem with sophisticated labeling is that current in-plane processor designs have very small amounts of memory, which makes it difficult to store learned convolutional masks or similar. Future processors for bottom-up processing may move beyond operating purely in the image plane, and use a 3D stacking approach which could be well suited to implementing the layers of a CNN. Currently 3D silicon stacks are hard to manufacture, and there are particular challenges around heat dissipation and cost, but the time will surely come when extracting a feature hierarchy is a built-in capability of a computer vision camera. Work such as *featuremetric* tracking [235] shows the wide general use this would have. We should investigate the full range of outputs that a single purely feed-forward CNN could produce when trained with a multi-task learning approach.

An interesting question is whether processing close to the image plane will remain

purely bottom-up, or whether two-way communication between cameras and the main processor will be worthwhile. This would enable model predictions from the world map to be delivered to the camera at some rate, and therefore for higher level processing to be carried out there such as model-based tracking of the camera's own motion or known objects.

In the limit, if a fully model-based prediction is able to be communicated to the camera, then the camera need only return information which is *different* from the prediction. This is in some sense a limit of the way that an event camera works. An event camera outputs data if a pixel changes in brightness — which is like assuming that the default is that the camera's view of the scene will stay the same. A general 'model-based event camera' will output data if something happens which differs from its prediction. These questions come down to key issues of 'bottom-up vs. top-down' processing, and we will consider them further in the next section.

As a final note here, any Spatial AI system must ultimately deliver a task-determined output. This could be the commands sent to robot actuators, communication to be sent to another robot or annotations and displays to be sent to a human operator in an IA setting. Just as 'close to the sensor' processing is efficient, there should also be a role for 'close to the actuator' processing, particularly because actuators or communication channels have their own types of sensing (torque feedback for actuators; perhaps eye tracking or other measures for an AR display) which need to be taken account of and fused in tight loops.

## 18.6 Convergent Distributed Computation with Gaussian Belief Propagation

We have argued that efficiency and low power demand a graph design for Spatial AI systems, where processing and local storage is divided into modules and spread potentially across several hardware processors and sensors, but so far said little about the algorithms that will keep the stored representations up to date. As explained throughout this handbook, probability and estimation theory provide the fundamental background for state estimation in SLAM, and in particular factor graphs [250] have been universally useful as a flexible representation of the structure of the complex structure of SLAM systems.

Sutton's famous 'Bitter Lesson' of AI [1060] is that 'general methods that leverage computation are ultimately the most effective, and by a large margin'. In Spatial AI, if we truly believe that low-power computation and storage resources will become more and more distributed, both within and across processors, the right algorithms to bet on should be ready to scale with these conditions. The purest representation of the knowledge in a Spatial AI problem is the factor graph itself, rather than probability distributions derived from it, which will always have to be stored with some approximation. What we are really seeking is an algorithm which implements Spatial AI by storing the factor graph as the master representation and operating

on it in place using local computation and message passing to implement estimation of variables as needed but taking account of global influence. Messages should continually ‘bubble’ around a large factor graph, which is changing continually with the addition of new measurement factors and variable nodes, and perhaps never reaching full convergence, but always being close in a way which can be controlled. It may be that estimation processes will proceed in an attention-driven way, using a lot of computation to bring high quality to currently important areas or aspects, which then are allowed to fade to a less up-to-date state once attention moves on, in a ‘just-in-time’ style.

If we wish estimation on factor graphs to have the properties of purely local computation and data storage, we must get away from the idea that a ‘god’s eye view’ of the whole structure of the graph will ever be available. We are guided towards methods where each node of a processing and storage graph can operate with minimum knowledge of the whole graph structure — at a minimum, only purely local information about itself and its near neighbors.

This is the character of the Belief Propagation (BP) family of algorithms for inference of probabilistic factor graphs. In BP, each variable and factor node processes messages with no knowledge about the rest of the graph other than its direct neighbors, and BP can converge with arbitrary, asynchronous message passing schedules which need no global coordination.

In particular, the most important techniques in current geometric Spatial AI estimation are all Gaussian-based techniques such as Extended Kalman Filtering and BA. Gaussian-based methods have very close links to linear algebra, because optimizing Gaussian likelihoods is equivalent to least-squares minimization which involves the solution of linear systems. Gaussian Belief Propagation (GBP) is the specific form of BP when all factors and variables in a graph have Gaussian form, and it has particularly strong properties of convergence to optimal solutions in loopy graphs.

Davison and Ortiz gave a general introduction to GBP in FutureMapping 2 [242], and an interactive tutorial is presented in the online article ‘A Visual Introduction to Gaussian Belief Propagation’ [830]. These articles showed that GBP is fully compatible with the non-linear and robust factors which are ubiquitous in Spatial AI problems, and can be applied to problems including PGO, sparse SLAM, surface fitting and image smoothing.

There had been a previous practical implementation of GBP to SLAM in [903], showing that it has some useful properties even when implemented on a single CPU, but Ortiz *et al.* [829] demonstrated its longer-term potential by using it to implement high performance BA on a Graphcore IPU chip, with a fully distributed implementation solving small real vision optimization problems 30X faster than on a CPU. Figure 18.2 shows the simple way that a BA factor graph was laid down on the tiles of a graph processor to take full advantage of its parallelism.

The generality of distributed computation on graphs with GBP was taken

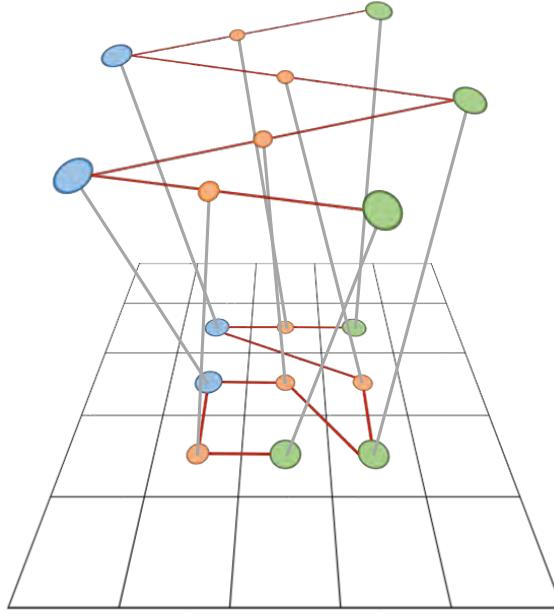


Figure 18.2 Mapping a bundle adjustment factor graph onto the tiles (cores) of a graph processor, allowing rapid, distributed, in-place inference via Gaussian Belief Propagation, from [829] (©IEEE). Here we display the most simple mapping in which each node in the factor graph is mapped onto a single arbitrary tile. Keyframe nodes are blue, landmark nodes are green and measurement factor nodes are orange. Tiles process synchronously, alternating compute and exchange steps.

even further by Murai *et al.* in Robot Web [785], showing that it can also be used for many-robot Spatial AI problems. Specifically, multi-robot localization was formulated as a single factor graph as shown in Figure 18.3, which was divided up and stored across a network of robots connected by ad-hoc communication such as Wi-Fi. Robot Web showed GBP can achieve convergent, accurate estimation even with asynchronous message passing and in the presence of failing sensors and dropped communication. These properties also promise much for single robot Spatial AI when we envision the very long-term prospects for robot processors, which may achieve even higher performance per unit energy by becoming more brain-like — with perhaps low-precision number representation and asynchronous or event-driven parallelism.

### 18.7 Continual Learning within Factor Graphs

We discussed the increasing importance of learned components in Spatial AI in Section 18.3, but interfacing these with modular state estimation continues to be

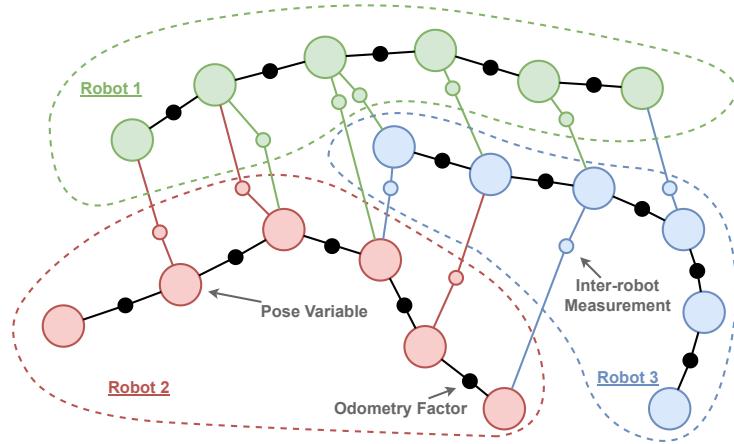


Figure 18.3 A factor graph for multi-robot localization, taken from Robot Web [785] (©IEEE). Responsibility for storing and updating it is divided up between the multiple robots participating, as shown by the colored regions separated by dotted lines. Each robot maintains its own pose variable nodes, odometry factors, and factors for the inter-robot measurements made by its sensors, and carries out continuous GBP on this graph fragment. Message passing across dotted line bound arises is via Robot Web Pages published and updated by each robot, and happens on an asynchronous and ad-hoc basis.

clunky and with many limitations. Looking further ahead, we believe that the following ambitious properties are important for learning in Spatial AI:

- 1 Learning will be a key part of Spatial AI systems, but should run seamlessly alongside existing tried-and-tested hand-designed probabilistic inference algorithms which can correctly weigh and combine multiple uncertain information sources.
- 2 The primary mode of learning should be self-supervised, meaning that a running system can learn useful abstractions without external help, such as from a human providing labelled datasets.
- 3 Learning in Spatial AI should be continual, taking place as part of a running system whose performance is gradually improving in consequence. We should move away from the current artificial divide between ‘training time’ and ‘test time’.
- 4 Learned components should ideally be interpretable, or human understandable, at least at some level of modules and how they connect together.

Most learning algorithms, and in particular standard neural networks, are trained and used separately. Training finds values for the parameters (weights) of a model by gradient descent using empirical numbers to determine learning rates and stop-

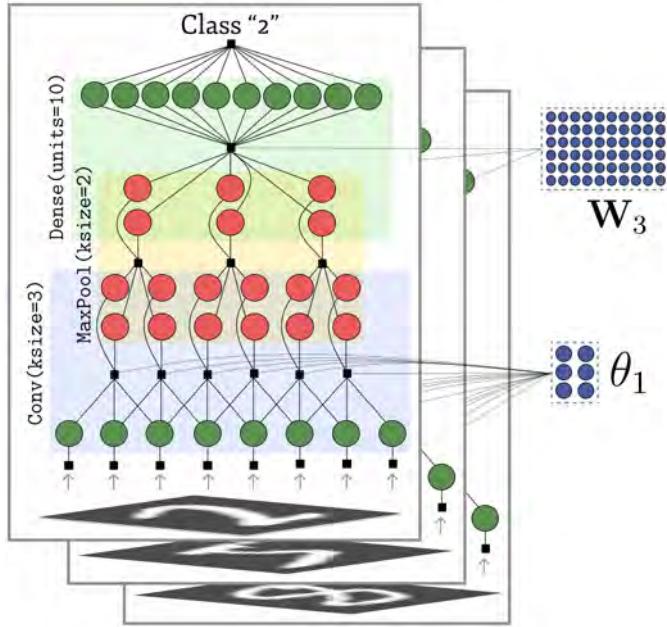


Figure 18.4 In GBP Learning [790], factor graphs are designed with structure which mirrors common NN architectures, enabling distributed training and prediction with GBP. Learnable **parameters** are included as random variables (circles), as are **inputs**, **outputs** and **activations**. The **parameters** are shared over across all observations, where the other variables are copied once per observation. Factors (black squares) between layers constrain their representations to be locally consistent, while those attached to inputs and outputs encourage compatibility with observation. The inter-layer factors are non-linear to enable soft-switching behavior. This example architecture for image classification comprises convolutional, max pooling and dense projection layers. The same architecture could be trained without supervision by removing the output observation factor.

ping criteria, and ad-hoc concepts are used such as dividing data into training and validation sets. It is very hard to imagine how such methods could be used properly for continual, self-supervised learning alongside probabilistic estimation.

We first argue that the points above lead us to employ a probabilistic approach to machine learning for Spatial AI. In probabilistic, or Bayesian ML, the parameters which represent a learned model are estimated using the same probabilistic inference laws as in standard probabilistic inference. There is no fundamental difference between state estimation in probabilistic inference, and model learning in Bayesian ML. It is simply a matter of structure and scale. Using the same machinery for both means that we can correctly attack self-supervised, continual learning alongside standard Bayesian inference.

Efficient and scalable Bayesian ML is a challenging topic, but Nabarro *et al.*

[790] have recently made a proposal and preliminary demonstrations of a GBP Learning approach which is fully compatible with the factor graph Spatial AI vision of this chapter. The key idea is that overparameterized learning structures are built into factor graphs themselves, and that learning takes place as Bayesian inference using standard Gaussian Belief Propagation machinery. Instead of the neurons and weights of a neural networks, we build a graph of variables and factors with a similar structure. Variables infer values in a manner equivalent to weight learning, but this can happen continuously and with no explicit difference between training and inference time. Non-linear factors play the role of neurons, able to soft-switch and therefore combine in varied patterns to represent complex functions. Figure 18.4 from [790] shows a factor graph block with convolutional structure which has been used in a system able to continual learning of image denoising and classification.

This approach or something similar could unlock general continual learning in Spatial AI. For instance, a robot could learn a smoothing or segmentation model for a scene while reconstructing it. The unified factor graph structure of both learning and estimations components of its systems would allow many choices of efficient or distributed estimation, or the easy synchronization of learned elements across multiple devices.

## 18.8 Performance Metrics

One of the hypotheses we made at the start of this chapter is that the usefulness of a Spatial AI system for a wide range of tasks is well represented by a relatively small number of performance measures which have general importance.

The most common focus in performance measurement in SLAM is on localization accuracy, and there have been several efforts to create benchmark datasets for this (e.g. [1042]). An external pose measurement from a motion capture system is considered as the ground truth against which a SLAM algorithm is compared.

We have argued that Spatial AI is about much more than pose estimation, and more recent datasets have tried to broaden the scope of what can be evaluated. Dense scene modelling is difficult to evaluate because it requires an expensive and time-consuming process such as detailed laser scanning to capture a complete model of a real scene which is accurate and complete enough to be considered ground truth. The alternative, for this and other axes of evaluation, is to generate synthetic test data using computer graphics, such as the ICL-NUIM dataset [429]. Recently this approach has been extended also to provide ground truth for semantic labeling (e.g., SceneNet RGB-D [743]), another output where it is difficult to get good ground truth for real data. It is natural to be suspicious of the value of evaluation against synthetic test data, and there are many new approaches to gathering large scale real mapping data, such as crowdsourcing (e.g. ScanNet [238]). However, synthetic data is getting better all the time and we believe will only grow in importance.

Stepping back to a bigger point, we can question the value of using benchmarks

at all for Spatial AI systems. It is an often heard comment that computer vision researchers are hooked on dataset evaluation, and that far too much effort has been spent on optimizing and combining algorithms to achieve a few more percentage points on benchmarks rather than working on new ideas and techniques. SLAM, due to its real-time nature, and wide range of useful outputs and performance levels for different applications, has been particularly difficult to capture by meaningful benchmarks. We have usually felt that more can be learned about the usefulness of a visual SLAM system by playing with it for a minute or so, adaptively and qualitatively checking its behavior via live visualisations, than from its measured performance against any benchmark available. Progress has therefore been more meaningfully represented by the progress of high quality real-time open source SLAM research systems (*e.g.*, MonoSLAM [244, 243], PTAM [582], KinectFusion [798], LSD-SLAM [307], ORB-SLAM [784], OKVIS [643], SVO [333], ElasticFusion [1181], DROID-SLAM [1079], Gaussian Splatting SLAM [737], MASt3R-SLAM [786]) that people can experiment with, rather than benchmarks.

Benchmarks for SLAM have been unsatisfactory because they make assumptions about the scene type and shape, camera and other sensor choices and placement, frame-rate and resolution, etc., and focus in on certain evaluation aspects such as accuracy while downplaying other arguably more important ones such as efficiency or robustness. For instance, many papers evaluating algorithms against accuracy benchmarks make choices among the test sequences available in a dataset such as [1042] and report performance only on those where they basically ‘work’.

This brings us to ask whether we should build benchmarks and aim to evaluate and compare SLAM systems at all? We still argue yes, but the focus on broadening what is meant by a benchmark for a Spatial AI system, and an acknowledgement that we should not put too much faith in what they tell us.

The SLAMBench framework released by the PAMELA research project [792] represented initial work on looking at the performance of a whole Spatial AI system. In SLAMBench, a SLAM algorithm (specifically KinectFusion [798]) is measured in terms of both accuracy and computational cost across a range of processor platforms and using different language implementations. Over the longer term, we believe that benchmarking should move towards measures which have the general ability to predict performance on tasks that a Spatial AI might need to perform. This will clearly be a multi-objective set of metrics, and analysis of Pareto fronts [793] will permit choices to be made for a particular application.

A possible set of metrics includes:

- 1 Local pose accuracy in newly explored areas (visual odometry drift rate).
- 2 Long-term metric pose repeatability in well mapped areas.
- 3 Tracking robustness percentage.
- 4 Relocalisation robustness percentage.
- 5 SLAM system latency.

- 6 Dense distance prediction accuracy at every pixel.
- 7 Object segmentation accuracy.
- 8 Object classification accuracy.
- 9 Pixel registration accuracy for augmented reality.
- 10 Scene change detection accuracy.
- 11 Power usage.
- 12 On-device data movement, measured in bits×millimetres.

### 18.9 Further Readings & Recent Trends

To conclude, the research area of Spatial AI and SLAM will continue to gain in importance, and evolve towards the general 3D perception capability needed for many different types of application with the full fruition of the combination of hand-designed estimation and machine learning techniques. However, wide use in real applications will require this advance in capability to be accompanied by driving down the resources required, and this needs joined-up thinking about algorithms, processors, and sensors. The key to efficient systems will be to identify their computational structure, and to design for sparse graph patterns in algorithms and data storage, and to fit this as closely as possible to the new types of hardware which will soon gain in importance.

Both hand-designed state estimation and machine learning will continue to be important, and must be fully integrated to enable flexible and continual learning. We believe we will witness an interesting convergence, as state estimation systems add more and more learned elements, and learned networks introduce increased domain knowledge and structure. Either route should lead to the same place, and the same research questions about computational structure will need to be tackled.

## Epilogue

As the final pages of this handbook turn, we, the authors, wish to leave you with some thoughts that have guided our journey. It is our sincere hope that this work will become more than just a guide, inspiring those who explore spatial understanding.

---

“Trust the math.”

“SLAM is not just about mapping spaces, it’s about enabling machines to perceive, understand, and reason about the world around them. Differentiable optimization will play a key role in this transformation, turning raw data into actionable spatial knowledge that can be learned, adapted, and generalized.”

“Learning the theory is important, but it’s applying it in robotic systems that truly consolidates the knowledge.”

“If someone tells you “SLAM is solved,” don’t listen to them. Focus on solving a problem that is 2 or 3 steps ahead of the state of the art. Your work will stand the test of time and be relevant for longer.”

“Always start with a simple toy problem (*i.e.*, an easy case with perfectly controlled synthetic data), and then iterate from this point on to more complex cases.”

“SLAM is about demos! Build some nice visualization graphics, plug in a live camera, and you’ll learn more about your system in 5 minutes than from any number of results on datasets.”

“Multimodality (sensory and probabilistic)”

“Having a clear notation is the best tool to spot modeling errors: do it with your math, your code, and your papers!”

“The ability to extract truth from a collection of conflicting sensor measurements is the most important thing in SLAM.”

“Everything seems impossible —until someone else accomplishes it. My motto is: “Everything is possible; you just need to think harder to make it happen”.”

“There are countless new sensors just waiting for robotic researchers to discover and apply them to fresh challenges —take event cameras, for example.”

“Abstract geometric thinking while remaining practical and experimenting with rigor and perseverance, but never alone, are the two main things that make you enjoy SLAM.”

“Research takes passion, technical skills, and (a sometimes misguided) optimism.”

“Robot perception is a mirror to look into our mind and a gateway to extending it into the world.”

“SLAM is the backbone of robot autonomy. Its rigorous mathematical formulation as an optimization problem over motion and sensing models holds tremendous potential to capture much more general settings, with suboptimality, efficiency, and uncertainty guarantees, than what is possible today. As a reader of this handbook, you are starting a mission to maintain the rigor of SLAM and to ensure that SLAM remains the interface for robot decision-making.”



## References

- [1] A, Zeng, Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., Lee, J., Vanhoucke, V., and Florence, P. 2022. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. *arXiv preprint*.
- [2] Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, et al. 2016. TensorFlow: a system for Large-Scale machine learning. Pages 265–283 of: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*.
- [3] Abate, M., Chang, Y., Hughes, N., and Carlone, L. 2023a. Kimera2: Robust and Accurate Metric-Semantic SLAM in the Real World. In: *Intl. Sym. on Experimental Robotics (ISER)*.
- [4] Abate, M., Schwartz, A., Wong, X.I., Luo, W., Littman, R., Klinger, M., Kuhnert, L., Blue, D., and Carlone, L. 2023b. Multi-Camera Visual-Inertial Simultaneous Localization and Mapping for Autonomous Valet Parking. In: *Intl. Sym. on Experimental Robotics (ISER)*. ,.
- [5] Achiam, Josh, Adler, Steven, Agarwal, Sandhini, Ahmad, Lama, Akkaya, Ilge, Aleman, Florencia Leoni, Almeida, Diogo, Altenschmidt, Janko, Altman, Sam, Anadkat, Shyamal, et al. 2023. Gpt-4 technical report. *arXiv preprint*.
- [6] Adolfsson, Daniel, Magnusson, Martin, Alhashimi, Anas, Lilienthal, Achim J., and Andreasson, Henrik. 2021. CFEAR Radarodometry – Conservative Filtering for Efficient and Accurate Radar Odometry. Pages 5462–5469 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [7] Adolfsson, Daniel, Castellano-Quero, Manuel, Magnusson, Martin, Lilienthal, Achim J., and Andreasson, Henrik. 2022. CorAl: Introspection for robust radar and lidar perception in diverse environments using differential entropy. *Robotics and Autonomous Systems*, May, 104136.
- [8] Adolfsson, Daniel, Magnusson, Martin, Alhashimi, Anas, Lilienthal, Achim J., and Andreasson, Henrik. 2023a. Lidar-Level Localization With Radar? The CFEAR Approach to Accurate, Fast, and Robust Large-Scale Radar Odometry in Diverse Environments. *IEEE Trans. Robotics*, **39**(2), 1476–1495.
- [9] Adolfsson, Daniel, Karlsson, Mattias, Kubelka, Vladimír, Magnusson, Martin, and Andreasson, Henrik. 2023b. TBV Radar SLAM – trust but verify loop candidates. *IEEE Robotics and Automation Letters*, **8**, 3613–3620.

- [10] Aftab, Khurram, and Hartley, Richard. 2015. Convergence of iteratively re-weighted least squares to robust m-estimators. Pages 480–487 of: *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE.
- [11] Agarwal, P., Grisetti, G., Tipaldi, G. D., Spinello, L., Burgard, W., and Stachniss, C. 2014. Experimental Analysis of Dynamic Covariance Scaling for Robust Map Optimization Under Bad Initial Estimates. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [12] Agarwal, Pratik, Tipaldi, Gian Diego, Spinello, Luciano, Stachniss, Cyrill, and Burgard, Wolfram. 2013. Robust map optimization using dynamic covariance scaling. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [13] Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. 2009. Building Rome in a Day. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [14] Agarwal, Sameer, Furukawa, Yasutaka, Snavely, Noah, Simon, Ian, Curless, Brian, Seitz, Steven M, and Szeliski, Richard. 2011. Building Rome in a day. *Communications of the ACM*, **54**(10), 105–112.
- [15] Agarwal, Sameer, Mierle, Keir, and Team, The Ceres Solver. 2022 (3). *Ceres Solver*.
- [16] Agate, M., Grimsdale, R. L., and Lister, P. F. 1991. The HERO Algorithm for Ray-tracing Octrees. Pages 61–73 of: *Advances in Computer Graphics Hardware IV*.
- [17] Agia, Christopher, Jatavallabhula, Krishna Murthy, Khodeir, Mohamed, Miksik, Ondrej, Vineet, Vibhav, Mukadam, Mustafa, Paull, Liam, and Shkurti, Florian. 2022. Taskography: Evaluating Robot Task Planning over Large 3D Scene Graphs. Pages 46–58 of: *Conf. on Robot Learning (CoRL)*.
- [18] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, Z. 2019. Differentiable Convex Optimization Layers. In: *Advances in Neural Information Processing Systems*.
- [19] Agrawal, Varun, Bertrand, Sylvain, Griffin, Robert J., and Dellaert, Frank. 2022. Proprioceptive State Estimation of Legged Robots with Kinematic Chain Modeling. Pages 178–185 of: *IEEE Intl. Conf. on Humanoid Robots*.
- [20] Agudo, Antonio, and Moreno-Noguer, Francesc. 2015. Simultaneous pose and non-rigid shape with particle dynamics. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [21] Agudo, Antonio, Agapito, Lourdes, Calvo, Begona, and Montiel, J.M.M. 2014. Good vibrations: A modal analysis approach for sequential non-rigid structure from motion. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [22] Agudo, Antonio, Moreno-Noguer, Francesc, Calvo, Begoña, and Montiel, José María Martínez. 2015. Sequential non-rigid structure from motion using physical priors. *IEEE Trans. Pattern Anal. Machine Intell.*, **38**(5), 979–994.
- [23] Akhter, Ijaz, Sheikh, Yaser, Khan, Sohaib, and Kanade, Takeo. 2011. Trajectory space: A dual representation for nonrigid structure from motion. *IEEE Trans. Pattern Anal. Machine Intell.*, **33**(7), 1442–1456.
- [24] Al-Rfou, Rami, Alain, Guillaume, Almahairi, Amjad, Angermueller, Christof, Bahdanau, Dzmitry, Ballas, Nicolas, Bastien, Frédéric, Bayer, Justin, Belikov, Anatoly, Belopolsky, Alexander, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint*, arXiv–1605.

- [25] Alhashimi, Anas, Adolfsson, Daniel, Andreasson, Henrik, Lilienthal, Achim J., and Magnusson, Martin. 2024. BFAR: improving radar odometry estimation using a bounded false alarm rate detector. *Autonomous Robots*, **48**(29).
- [26] Alismail, Hatem, Browning, Brett, and Lucey, Simon. 2016. Photometric Bundle Adjustment for Vision-Based SLAM. Pages 324–341 of: *Asian Conf. on Computer Vision (ACCV)*.
- [27] Alizadeh, F., Haeberly, J.P.A., and Overton, M.L. 1997. Complementarity and nondegeneracy in semidefinite programming. *Mathematical Programming*, **77**, 111–128.
- [28] Ambruş, Rareş, Bore, Nils, Folkesson, John, and Jensfelt, Patric. 2014. Meta-rooms: Building and maintaining long term spatial models in a dynamic world. Pages 1854–1861 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [29] Amestoy, P.R., Davis, T., and Duff, I.S. 1996. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, **17**(4), 886–905.
- [30] Amini, Alexander, Wang, Tsun-Hsuan, Gilitschenski, Igor, Schwarting, Wilko, Liu, Zhijian, Han, Song, Karaman, Sertac, and Rus, Daniela. 2022. VISTA 2.0: An Open, Data-driven Simulator for Multimodal Sensing and Policy Learning for Autonomous Vehicles. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [31] Amir, Arnon, Taba, Brian, Berg, David, Melano, Timothy, McKinstry, Jeffrey, Nolfo, Carmelo Di, Nayak, Tapan, Andreopoulos, Alexander, Garreau, Guillaume, Mendoza, Marcela, Kusnitz, Jeff, Debole, Michael, Esser, Steve, Delbruck, Tobi, Flickner, Myron, and Modha, Dharmendra. 2017. A Low Power, Fully Event-Based Gesture Recognition System. Pages 7388–7397 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [32] Andersson, Joel AE, Gillis, Joris, Horn, Greg, Rawlings, James B, and Diehl, Moritz. 2019. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, **11**(1), 1–36.
- [33] Ankenbauer, Jacqueline, Lusk, Parker C., and How, Jonathan P. 2023 (Mar.). *Global Localization in Unstructured Environments Using Semantic Object Maps Built from Various Viewpoints*.
- [34] Antonante, P., Tzoumas, V., Yang, H., and Carlone, L. 2021. Outlier-Robust Estimation: Hardness, Minimally Tuned Algorithms, and Applications. *IEEE Trans. Robotics*, **38**(1), 281–301. .
- [35] Arandjelovic, Relja, Gronat, Petr, Torii, Akihiko, Pajdla, Tomas, and Sivic, Josef. 2016. NetVLAD: CNN architecture for weakly supervised place recognition. Pages 5297–5307 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [36] Armeni, Iro, He, Zhi-Yang, Gwak, JunYoung, Zamir, Amir R, Fischer, Martin, Malik, Jitendra, and Savarese, Silvio. 2019. 3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera. Pages 5664–5673 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [37] Asgharivaskasi, Arash, and Atanasov, Nikolay. 2023. Semantic OcTree Mapping and Shannon Mutual Information Computation for Robot Exploration. *IEEE Trans. Robotics*, **39**(3), 1910–1928.

- [38] Asgharivaskasi, Arash, Girke, Fritz, and Atanasov, Nikolay. 2025. Riemannian Optimization for Active Mapping With Robot Teams. *IEEE Trans. Robotics*, **41**, 1077–1097.
- [39] Asokan, Anju, and Anitha, JJESI. 2019. Change detection techniques for remote sensing applications: A survey. *Earth Science Informatics*, **12**, 143–160.
- [40] Atanasov, Nikolay, Zhu, Menglong, Daniilidis, Kostas, and Pappas, George J. 2014. Semantic Localization Via the Matrix Permanent. In: *Robotics: Science and Systems (RSS)*.
- [41] Atanasov, Nikolay, Zhu, Menglong, Daniilidis, Kostas, and Pappas, George J. 2016. Localization from semantic observations via the matrix permanent. *Intl. J. of Robotics Research*, **35**(1-3), 73–99.
- [42] Bae, Gwangtak, Choi, Changwoon, Heo, Hyeongjun, Kim, Sang Min, and Kim, Young Min. 2024. I2-SLAM: Inverting Imaging Process for Robust Photorealistic Dense SLAM. In: *European Conf. on Computer Vision (ECCV)*.
- [43] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. 2015. Neural machine translation by jointly learning to align and translate. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [44] Ballester, Irene, Fontán, Alejandro, Civera, Javier, Strobl, Klaus H, and Triebel, Rudolph. 2021. DOT: Dynamic object tracking for visual SLAM. Pages 11705–11711 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [45] Banani, M. El, Raj, A., Maninis, K., Kar, A., Li, Y., Rubinstein, M., Sun, D., Guibas, L., Johnson, J., and Jampani, V. 2024. Probing the 3D Awareness of Visual Foundation Models. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [46] Barath, Daniel, Noskova, Jana, Ivashechkin, Maksym, and Matas, Jiri. 2020. MAGSAC++, a fast, reliable and accurate robust estimator. Pages 1304–1312 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [47] Barfoot, T D. 2024. *State Estimation for Robotics*. 2nd edn. Cambridge University Press.
- [48] Barfoot, T. D., and Furgale, P. T. 2014. Associating Uncertainty with Three-Dimensional Poses for use in Estimation Problems. *IEEE Trans. Robotics*, **30**(3), 679–693.
- [49] Barfoot, T D, Forbes, J R, and D’Eleuterio, G M T. 2022. Vectorial Parameterizations of Pose. *Robotica*, **40**(7), 2409–2427.
- [50] Barfoot, Tim D, Tong, Chi Hay, and Särkkä, Simo. 2014. Batch Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression. Pages 1–10 of: *Robotics: Science and Systems (RSS)*, vol. 10. Citeseer.
- [51] Barfoot, Timothy D. 2017. *State estimation for robotics*. Cambridge University Press.
- [52] Barnes, Dan, and Posner, Ingmar. 2020. Under the Radar: Learning to Predict Robust Keypoints for Odometry Estimation and Metric Localisation in Radar. Pages 9484–9490 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [53] Barnes, Dan, Weston, Rob, and Posner, Ingmar. 2019. Masking by Moving: Learning Distraction-Free Radar Odometry from Pose Information. In: *Conf. on Robot Learning (CoRL)*.

- [54] Barnes, Dan, Gadd, Matthew, Murcutt, Paul, Newman, Paul, and Posner, Ingmar. 2020. The Oxford radar robotcar dataset: A radar extension to the Oxford robotcar dataset. Pages 6433–6438 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [55] Barranco, Francisco, Fermüller, Cornelia, Aloimonos, Yiannis, and Delbrück, Tobi. 2016. A Dataset for Visual Navigation with Neuromorphic Methods. *Front. Neurosci.*, **10**, 49.
- [56] Barrau, A., and Bonnabel, S. 2017. The Invariant Extended Kalman Filter as a Stable Observer. *IEEE Trans. on Automatic Control*, **62**(4), 1797–1812.
- [57] Barron, Jonathan T. 2019. A general and adaptive robust loss function. Pages 4331–4339 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [58] Barron, Jonathan T., Mildenhall, Ben, Verbin, Dor, Srinivasan, Pratul P., and Hedman, Peter. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [59] Bartoli, Adrien, Gérard, Yan, Chadebecq, François, Collins, Toby, and Pizarro, Daniel. 2015. Shape-from-template. *IEEE Trans. Pattern Anal. Machine Intell.*, **37**(10), 2099–2118.
- [60] Bauernfeind, Carl Max v. 1856. *Elemente der Vermessungskunde: ein Lehrbuch der praktischen Geometrie*. Cotta.
- [61] Bavle, Hriday, Sanchez-Lopez, Jose Luis, Shaheer, Muhammad, Civera, Javier, and Voos, Holger. 2022. Situational Graphs for Robot Navigation in Structured Indoor Environments. *IEEE Robotics and Automation Letters*, **7**(4), 9107–9114.
- [62] Bay, Herbert, Tuytelaars, Tinne, and Van Gool, Luc. 2006. Surf: Speeded up robust features. Pages 404–417 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [63] Bay, Herbert, Ess, Andreas, Tuytelaars, Tinne, and Van Gool, Luc. 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding (CVIU)*, **110**(3), 346–359.
- [64] Bazin, J.C., Seo, Y., Hartley, R.I., and Pollefeys, M. 2014. Globally optimal inlier set maximization with unknown rotation and focal length. Pages 803–817 of: *European Conf. on Computer Vision (ECCV)*.
- [65] Beeson, Patrick, Modayil, Joseph, and Kuipers, Benjamin. 2010. Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy. *Intl. J. of Robotics Research*, **29**(4), 428–459.
- [66] Behley, J., and Stachniss, C. 2018. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In: *Robotics: Science and Systems (RSS)*.
- [67] Behley, J., Steinhage, V., and Cremers, A. B. 2012. Performance of Histogram Descriptors for the Classification of 3D Laser Range Data in Urban Environments. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [68] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. 2019. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [69] Behley, J., Milioto, A., and Stachniss, C. 2021. A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [70] Behley, Jens, Steinhage, Volker, and Cremers, Armin B. 2015. Efficient Radius Neighbor Search in Three-dimensional Point Clouds. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [71] Bennett, M. S. 2024. *A Brief History of Intelligence*. Harper Collins.
- [72] Bentley, J.L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, **18**(9), 509–517.
- [73] Berg, Matthew, Konidaris, George, and Tellex, Stefanie. 2022. Using Language to Generate State Abstractions for Long-Range Planning in Outdoor Environments. Pages 1888–1895 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [74] Bernreiter, Lukas, Gawel, Abel, Sommer, Hannes, Nieto, Juan, Siegwart, Roland, and Lerma, Cesar Cadena. 2019. Multiple Hypothesis Semantic Mapping for Robust Data Association. *IEEE Robotics and Automation Letters*, **4**(4), 3255–3262.
- [75] Bescós, Berta, Civera, Javier, and Neira, José. 2017. Removing dynamic objects from 3d maps using geometry and learning. In: *IEEE Conf. on Intelligent Robots and Systems (IROS) workshop of learning for localization and mapping*.
- [76] Bescós, Berta, Campos, Carlos, Tardós, Juan D., and Neira, José. 2021. DynaSLAM II: Tightly-Coupled Multi-Object Tracking and SLAM. *IEEE Robotics and Automation Letters*, **6**(3), 5191–5198.
- [77] Bescós, Berta, Cadena, César, and Neira, José. 2021. Empty Cities: A Dynamic-Object-Invariant Space for Visual SLAM. *IEEE Trans. Robotics*, **37**(2), 433–451.
- [78] Besl, Paul J., and McKay, Neil D. 1992a. Method for registration of 3-D shapes. Pages 586–606 of: *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie.
- [79] Besl, Paul J., and McKay, Neil D. 1992b. Method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Machine Intell.*, **14**(2), 239–256.
- [80] Bezdek, James C., and Hathaway, Richard J. 2003. Convergence of alternating optimization. *Neural, Parallel & Scientific Computations*, **11**(4), 351–368.
- [81] Bhardwaj, Mohak, Boots, Byron, and Mukadam, Mustafa. 2020. Differentiable gaussian process motion planning. Pages 10598–10604 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [82] Biber, Peter, and Straßer, Wolfgang. 2003. The normal distributions transform: A new approach to laser scan matching. Pages 2743–2748 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE.
- [83] Bierman, G.J. 1977. *Factorization methods for discrete sequential estimation*. Mathematics in Science and Engineering, vol. 128. New York: Academic Press.
- [84] Black, Kevin, Brown, Noah, Driess, Danny, Esmail, Adnan, Equi, Michael, Finn, Chelsea, Fusai, Niccolo, Groom, Lachy, Hausman, Karol, Ichter, Brian, Jakubczak, Szymon, Jones, Tim, Ke, Liyiming, Levine, Sergey, Li-Bell, Adrian, Mothukuri, Mohith, Nair, Suraj, Pertsch, Karl, Shi, Lucy Xiaoyang, Tanner, James, Vuong, Quan, Walling, Anna, Wang, Haohuan, and Zhilinsky, Ury. 2024.  $\pi_0$ : *A Vision-Language-Action Flow Model for General Robot Control*.

- [85] Black, Michael J., and Rangarajan, Anand. 1996. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Intl. J. of Computer Vision*, **19**(1), 57–91.
- [86] Blake, Andrew, and Zisserman, Andrew. 1987. *Visual reconstruction*. MIT Press.
- [87] Blanco, Jose-Luis. 2010. A tutorial on SE(3) transformation parameterizations and on-manifold optimization. *University of Malaga, Technical Report*, **3**, 6.
- [88] Bloesch, Michael, Hutter, Marco, Hoepflinger, Mark, Leutenegger, Stefan, Gehring, Christian, Remy, C. David, and Siegwart, Roland. 2012. State Estimation for Legged Robots - Consistent Fusion of Leg Kinematics and IMU. In: *Robotics: Science and Systems (RSS)*.
- [89] Bloesch, Michael, Gehring, Christian, Fankhauser, Péter, Hutter, Marco, Hoepflinger, Mark A., and Siegwart, Roland. 2013. State estimation for legged robots on unstable and slippery terrain. Pages 6058–6064 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [90] Bloesch, Michael, Omari, Sammy, Hutter, Marco, and Siegwart, Roland. 2015. Robust visual inertial odometry using a direct EKF-based approach. Pages 298–304 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [91] Bloesch, Michael, Czarnowski, Jan, Clark, Ronald, Leutenegger, Stefan, and Davison, Andrew J. 2018a. CodeSLAM—learning a compact, optimisable representation for dense visual SLAM. Pages 2560–2568 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [92] Bloesch, Michael, Burri, Michael, Sommer, Hannes, Siegwart, Roland, and Hutter, Marco. 2018b. The Two-State Implicit Filter Recursive Estimation for Mobile Robots. *IEEE Robotics and Automation Letters*, **3**(1), 573–580.
- [93] Bodlaender, Hans L. 2006. Treewidth: Characterizations, Applications, and Computations. Pages 1–14 of: *Graph-Theoretic Concepts in Computer Science*. Springer Berlin Heidelberg.
- [94] Bohren, Jonathan, Foote, Tully, Keller, Jim, Kushleyev, Alex, Lee, Daniel, Stewart, Alex, Vernaza, Paul, Derenick, Jason, Spletzer, John, and Satterfield, Brian. 2008. Little Ben: The Ben Franklin Racing Team’s entry in the 2007 DARPA Urban Challenge. *J. of Field Robotics*, **25**(9), 598–614.
- [95] Bonnabel, Silvère, and Barrau, Axel. 2015. An intrinsic Cramér-Rao bound on Lie groups. Pages 664–672 of: *International Conference on Geometric Science of Information*. Springer.
- [96] Bore, Nils, Ekekrantz, Johan, Jensfelt, Patric, and Folkesson, John. 2018. Detection and tracking of general movable objects in large three-dimensional maps. *IEEE Trans. Robotics*, **35**(1), 231–247.
- [97] Borrmann, Dorit, Nüchter, Andreas, Dakulović, Marija, Maurović, Ivan, Petrović, Ivan, Osmanković, Dinko, and Velagić, Jasmin. 2014. A mobile robot based system for fully automated thermal 3D mapping. *Advanced Engineering Informatics*, **28**(4), 425–440.
- [98] Borts, David, Liang, Erich, Broedermann, Tim, Ramazzina, Andrea, Walz, Stefanie, Palladin, Edoardo, Sun, Jipeng, Brueggemann, David, Sakaridis, Christos, Van Gool, Luc, Bijelic, Mario, and Heide, Felix. 2024. Radar Fields: Frequency-Space Neural Scene Representations for FMCW Radar.

- In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*. SIGGRAPH '24. New York, NY, USA: Association for Computing Machinery.
- [99] Bosse, M., Agamennoni, G., and Gilitschenski, I. 2016. Robust Estimation and Applications in Robotics. *Foundations and Trends in Robotics*, **4**(4), 225–269.
  - [100] Bosse, Michael, and Zlot, Robert. 2009. Continuous 3D scan-matching with a spinning 2D laser. Pages 4312–4319 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [101] Bosse, Michael, and Zlot, Robert. 2013. Place recognition using keypoint voting in large 3D lidar datasets. Pages 2677–2684 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
  - [102] Bosse, Michael, Zlot, Robert, and Flick, Paul. 2012. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *IEEE Trans. Robotics*, **28**(5), 1104–1119.
  - [103] Botsch, M., Spernatt, M., and Kobbelt, L. 2004. Phong splatting. In: *Symposium on Point-Based Graphics (PBG)*.
  - [104] Botsch, Mario, Hornung, Alexander, Zwicker, Matthias, and Kobbelt, Leif. 2005. High-Quality Surface Splatting on Today's GPUs. In: *Symposium on Point-Based Graphics (PBG)*.
  - [105] Boumal, Nicolas. 2013. On intrinsic Cramér-Rao bounds for Riemannian submanifolds and quotient manifolds. *IEEE Trans. Signal Processing*, **61**(7), 1809–1821.
  - [106] Boumal, Nicolas. 2023. *An introduction to optimization on smooth manifolds*.
  - [107] Boumal, Nicolas, Voroninski, Vladislav, and Bandeira, Afonso S. 2016. The non-convex Burer-Monteiro approach works on smooth semidefinite programs. Pages 2757–2765 of: *Conf. Neural Information Processing Systems (NIPS)*.
  - [108] Bowman, S.L., Atanasov, N., Daniilidis, K., and Pappas, G.J. 2017. Probabilistic data association for semantic SLAM. Pages 1722–1729 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [109] Boyle, Michael. 2017. The Integration of Angular Velocity. *Advances in Applied Clifford Algebras*, **27**(3), 2345–2374.
  - [110] Bradbury, James, Frostig, Roy, Hawkins, Peter, Johnson, Matthew James, Leary, Chris, Maclaurin, Dougal, Necula, George, Paszke, Adam, Vander-Plas, Jake, Wanderman-Milne, Skye, and Zhang, Qiao. 2018. *JAX: composable transformations of Python+NumPy programs*.
  - [111] Brandli, Christian, Berner, Raphael, Yang, Minhao, Liu, Shih-Chii, and Delbrück, Tobi. 2014. A 240x180 130dB 3 $\mu$ s Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE J. Solid-State Circuits*, **49**(10), 2333–2341.
  - [112] Brandão, Martim, Aladag, Omer Burak, and Havoutis, Ioannis. 2020. GaitMesh: Controller-Aware Navigation Meshes for Long-Range Legged locomotion Planning in Multi-Layered Environments. *IEEE Robotics and Automation Letters*, **5**(2), 3596–3603.
  - [113] Bregler, Christoph, Hertzmann, Aaron, and Biermann, Henning. 2000. Recovering non-rigid 3D shape from image streams. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [114] Briales, J., and Gonzalez-Jimenez, J. 2016 (Oct). Fast global optimality verification in 3D SLAM. Pages 4630–4636 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [115] Briales, Jesus, and Gonzalez-Jimenez, Javier. 2017. Convex Global 3D Registration with Lagrangian Duality. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [116] Briales, Jesus, Kneip, Laurent, and Gonzalez-Jimenez, Javier. 2018. A Certifiably Globally Optimal Solution to the Non-Minimal Relative Pose Problem. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [117] Brogan, William L. 1991. *Modern Control Theory*. Upper Saddle River, NJ: Prentice Hall.
- [118] Brohan, Anthony, Brown, Noah, Carbajal, Justice, Chebotar, Yevgen, Dabis, Joseph, Finn, Chelsea, Gopalakrishnan, Keerthana, Hausman, Karol, Herzog, Alex, Hsu, Jasmine, Ibarz, Julian, Ichter, Brian, Irpan, Alex, Jackson, Tomas, Jesmonth, Sally, Joshi, Nikhil, Julian, Ryan, Kalashnikov, Dmitry, Kuang, Yuheng, Leal, Isabel, Lee, Kuang-Huei, Levine, Sergey, Lu, Yao, Malla, Utsav, Manjunath, Deeksha, Mordatch, Igor, Nachum, Ofir, Parada, Carolina, Peralta, Jodilyn, Perez, Emily, Pertsch, Karl, Quiambao, Jornell, Rao, Kanishka, Ryoo, Michael, Salazar, Grecia, Sanketi, Pannag, Sayed, Kevin, Singh, Jaspiar, Sontakke, Sumedh, Stone, Austin, Tan, Clayton, Tran, Huong, Vanhoucke, Vincent, Vega, Steve, Vuong, Quan, Xia, Fei, Xiao, Ted, Xu, Peng, Xu, Sichun, Yu, Tianhe, and Zitkovich, Brianna. 2022. RT-1: Robotics Transformer for Real-World Control at Scale. In: *arXiv preprint*.
- [119] Brohan, Anthony, Brown, Noah, Carbajal, Justice, Chebotar, Yevgen, Chen, Xi, Choromanski, Krzysztof, Ding, Tianli, Driess, Danny, Dubey, Avinava, Finn, Chelsea, Florence, Pete, Fu, Chuyuan, Arenas, Montse Gonzalez, Gopalakrishnan, Keerthana, Han, Kehang, Hausman, Karol, Herzog, Alex, Hsu, Jasmine, Ichter, Brian, Irpan, Alex, Joshi, Nikhil, Julian, Ryan, Kalashnikov, Dmitry, Kuang, Yuheng, Leal, Isabel, Lee, Lisa, Lee, Tsang-Wei Edward, Levine, Sergey, Lu, Yao, Michalewski, Henryk, Mordatch, Igor, Pertsch, Karl, Rao, Kanishka, Reymann, Krista, Ryoo, Michael, Salazar, Grecia, Sanketi, Pannag, Sermanet, Pierre, Singh, Jaspiar, Singh, Anikait, Soricut, Radu, Tran, Huong, Vanhoucke, Vincent, Vuong, Quan, Wahid, Ayzaan, Welker, Stefan, Wohlhart, Paul, Wu, Jialin, Xia, Fei, Xiao, Ted, Xu, Peng, Xu, Sichun, Yu, Tianhe, and Zitkovich, Brianna. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In: *arXiv preprint*.
- [120] Brossard, Martin, Barrau, Axel, Chauchat, Paul, and Bonnabel, Silvère. 2022. Associating Uncertainty to Extended Poses for on Lie Group IMU Preintegration With Rotating Earth. *IEEE Trans. Robotics*, **38**(2), 998–1015.
- [121] Brune, Marvin, Meisen, Tobias, and Pomp, André. 2024. Survey of Deep Learning-Based Methods for FMCW Radar Odometry and Ego-Localization. *Applied Sciences*, **14**(6).
- [122] Brynte, Lucas, Larsson, Viktor, Iglesias, José Pedro, Olsson, Carl, and Kahl, Fredrik. 2021. On the Tightness of Semidefinite Relaxations for Rotation Estimation. *Journal of Mathematical Imaging and Vision*, **64**(1), 57–67.
- [123] Buchanan, Russell, Agrawal, Varun, Camurri, Marco, Dellaert, Frank, and Fallon, Maurice. 2023. Deep IMU Bias Inference for Robust Visual-Inertial

- Odometry With Factor Graphs. *IEEE Robotics and Automation Letters*, **8**(1), 41–48.
- [124] Burer, S., and Monteiro, R. 2004. Local Minima and Convergence in Low-Rank Semidefinite Programming. *Mathematical Programming*, **103**(3), 427–444.
- [125] Burer, Samuel, and Monteiro, Renato. 2003. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, **95**(2), 329–357.
- [126] Burner, Levi, Mitrokhin, Anton, Fermüller, Cornelia, and Aloimonos, Yianis. 2022. EVIMO2: An Event Camera Dataset for Motion Segmentation, Optical Flow, Structure from Motion, and Visual Inertial Odometry in Indoor Scenes with Monocular or Stereo Algorithms. *arXiv preprint*, May.
- [127] Burnett, Keenan, Wu, Yuchen, Yoon, David J., Schoellig, Angela P., and Barfoot, Timothy D. 2022. Are We Ready for Radar to Replace Lidar in All-Weather Mapping and Localization? *IEEE Robotics and Automation Letters*, **7**(4), 10328–10335.
- [128] Burnett, Keenan, Yoon, David J., Wu, Yuchen, Li, Andrew Zou, Zhang, Haowei, Lu, Shichen, Qian, Jingxing, Tseng, Wei-Kang, Lambert, Andrew, Leung, Keith YK, Schoellig, Angela P., and Barfoot, Timothy D. 2023. Boreas: A Multi-Season Autonomous Driving Dataset. *Intl. J. of Robotics Research*, **42**(12), 33–42.
- [129] Burnett, Keenan, Schoellig, Angela P., and Barfoot, Timothy D. 2024. Continuous-Time Radar-Inertial and Lidar-Inertial Odometry using a Gaussian Process Motion Prior.
- [130] Burnett, Keenan, Schoellig, Angela P., and Barfoot, Timothy D. 2025. IMU as an input versus a measurement of the state in inertial-aided state estimation. *Robotica*, 1–21.
- [131] Burri, Michael, Nikolic, Janosch, Gohl, Pascal, Schneider, Thomas, Rehder, Joern, Omari, Sammy, Achtelik, Markus W., and Siegwart, Roland. 2016. The EuRoC micro aerial vehicle datasets. *Intl. J. of Robotics Research*, **35**(10), 1157–1163.
- [132] Bustos, Á. P., and Chin, T. J. 2018. Guaranteed outlier removal for point cloud registration with correspondences. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(12), 2868–2882.
- [133] Bustos, Alvaro Parra, Chin, Tat-Jun, Neumann, Frank, Friedrich, Tobias, and Katzmann, Maximilian. 2019. A Practical Maximum Clique Algorithm for Matching with Pairwise Constraints. *arXiv preprint*.
- [134] Bylow, Erik, Sturm, Jürgen, Kerl, Christian, Kahl, Fredrik, and Cremers, Daniel. 2013. Real-time camera tracking and 3D reconstruction using signed distance functions. Page 2 of: *Robotics: Science and Systems (RSS)*, vol. 2.
- [135] Cabon, Yohann, Stoffl, Lucas, Antsfeld, Leonid, Csurka, Gabriela, Chidlovskii, Boris, Revaud, Jérôme, and Leroy, Vincent. 2025. *MUST3R: Multi-view Network for Stereo 3D Reconstruction*.
- [136] Cadena, Cesar, Carlone, Luca, Carrillo, Henry, Latif, Yasir, Scaramuzza, Davide, Neira, José, Reid, Ian, and Leonard, John J. 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, **32**(6), 1309–1332.
- [137] Caesar, Holger, Bankiti, Varun, Lang, Alex H, Vora, Sourabh, Lioung, Venice Erin, Xu, Qiang, Krishnan, Anush, Pan, Yu, Baldan, Giancarlo, and

- Beijbom, Oscar. 2020. nuscenes: A multimodal dataset for autonomous driving. Pages 11621–11631 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [138] Cai, Hongrui, Feng, Wanquan, Feng, Xuetao, Wang, Yan, and Zhang, Juyong. 2022a. Neural Surface Reconstruction of Dynamic Scenes with Monocular RGB-D Camera. In: *Advances in Neural Information Processing Systems (NIPS)*.
- [139] Cai, Kaiwen, Wang, Bing, and Lu, Chris Xiaoxuan. 2022b. AutoPlace: Robust Place Recognition with Single-chip Automotive Radar. Pages 2222–2228 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [140] Callmer, Jonas, Törnqvist, David, Gustafsson, Fredrik, Svensson, Henrik, and Carlblom, Pelle. 2011. Radar SLAM using visual features. *EURASIP Journal on Advances in Signal Processing*, **2011**(09), 1–11.
- [141] Calonder, Michael, Lepetit, Vincent, Strecha, Christoph, and Fua, Pascal. 2010. BRIF: Binary Robust Independent Elementary Features. Pages 778–792 of: *European Conference on Computer Vision (ECCV)*.
- [142] Campos, Carlos, Elvira, Richard, Rodríguez, Juan J Gómez, Montiel, José MM, and Tardós, Juan D. 2021. Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam. *IEEE Trans. Robotics*, **37**(6), 1874–1890.
- [143] Camurri, Marco, Fallon, Maurice, Bazeille, Stéphane, Radulescu, Andreea, Barasuol, Victor, Caldwell, Darwin G., and Semini, Claudio. 2017. Probabilistic Contact Estimation and Impact Detection for State Estimation of Quadruped Robots. *IEEE Robotics and Automation Letters*, **2**(2), 1023–1030.
- [144] Camurri, Marco, Ramezani, Milad, Nobili, Simona, and Fallon, Maurice. 2020. Pronto: A Multi-Sensor State Estimator for Legged Robots in Real-World Scenarios. *Frontiers in Robotics and AI*, **7**.
- [145] Cao, Shaozu, Lu, Xiuyuan, and Shen, Shaojie. 2022. GVINS: Tightly Coupled GNSS–Visual–Inertial Fusion for Smooth and Consistent State Estimation. *IEEE Trans. Robotics*, **38**(4), 2004–2021.
- [146] Carion, Nicolas, Massa, Francisco, Synnaeve, Gabriel, Usunier, Nicolas, Kirillov, Alexander, and Zagoruyko, Sergey. 2020. End-to-End Object Detection with Transformers. In: *European Conf. on Computer Vision (ECCV)*.
- [147] Carbone, L. 2023. Estimation Contracts for Outlier-Robust Geometric Perception. *Foundations and Trends (FnT) in Robotics, arXiv preprint: 2208.10521*.
- [148] Carbone, L., and Censi, A. 2014. From Angular Manifolds to the Integer Lattice: Guaranteed Orientation Estimation With Application to Pose Graph Optimization. *IEEE Trans. Robotics*, **30**(2), 475–492.
- [149] Carbone, L., and Dellaert, F. 2015. Duality-based Verification Techniques for 2D SLAM. Pages 4589–4596 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [150] Carbone, L., and Karaman, S. 2018. Attention and Anticipation in Fast Visual-Inertial Navigation. *IEEE Trans. Robotics*.
- [151] Carbone, L., Rosen, D.M., Calafiore, G.C., Leonard, J.J., and Dellaert, F. 2015a. Lagrangian Duality in 3D SLAM: Verification Techniques and Optimal Solutions. Pages 125–132 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

- [152] Carlone, L., Calafiore, G., and Dellaert, F. 2015b. Pose Graph Optimization in the Complex Domain: Lagrangian Duality and Optimal Solutions. In: *Robotics: Science and Systems Workshop*.
- [153] Carlone, Luca, and Calafiore, Giuseppe C. 2018. Convex relaxations for pose graph optimization with outliers. *IEEE Robotics and Automation Letters*, **3**(2), 1160–1167.
- [154] Caron, Mathilde, Touvron, Hugo, Misra, Ishan, Jégou, Hervé, Mairal, Julien, Bojanowski, Piotr, and Joulin, Armand. 2021. Emerging properties in self-supervised vision transformers. Pages 9650–9660 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [155] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. 2001. Reconstruction and representation of 3D objects with radial basis functions. Pages 67–76 of: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*. Association for Computing Machinery.
- [156] Cattaneo, Daniele, Vaghi, Matteo, and Valada, Abhinav. 2022. Lcdnet: Deep loop closure detection and point cloud registration for lidar slam. *IEEE Trans. Robotics*, **38**(4), 2074–2093.
- [157] Cazals, Frédéric, and Giesen, Joachim. 2006. *Delaunay Triangulation Based Surface Reconstruction*. Springer Berlin Heidelberg. Pages 231–276.
- [158] Cen, Sarah H., and Newman, Paul. 2018. Precise Ego-Motion Estimation with Millimeter-Wave Radar Under Diverse and Challenging Conditions. Pages 6045–6052 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [159] Censi, A. 2007. An accurate closed-form estimate of ICP’s covariance. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [160] Censi, Andrea, and Scaramuzza, Davide. 2014. Low-Latency Event-Based Visual Odometry. Pages 703–710 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [161] Chadwick, Jeffrey N., and Bindel, David S. 2015. An efficient solver for sparse linear systems based on rank-structured Cholesky factorization. *arXiv preprint*.
- [162] Chakravarthi, Bharatesh, Verma, Aayush Atul, Daniilidis, Kostas, Fermüller, Cornelia, and Yang, Yezhou. 2024. Recent Event Camera Innovations: A Survey. In: *European Conf. on Computer Vision Workshops*.
- [163] Chaney, Kenneth, Cladera, Fernando, Wang, Ziyun, Bisulco, Anthony, Hsieh, M. Ani, Korpela, Christopher, Kumar, Vijay, Taylor, Camillo J., and Daniilidis, Kostas. 2023. M3ED: Multi-Robot, Multi-Sensor, Multi-Environment Event Dataset. Pages 4016–4023 of: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops*.
- [164] Chang, Haonan, Boyalakuntla, Kowndinya, Lu, Shiyang, Cai, Siwei, Jing, Eric Pu, Keskar, Shreesh, Geng, Shijie, Abbas, Adeeb, Zhou, Lifeng, Bekris, Kostas, and Boualiouss, Abdeslam. 2023a. Context-Aware Entity Grounding with Open-Vocabulary 3D Scene Graphs. In: *Conf. on Robot Learning (CoRL)*.
- [165] Chang, Y., Ebadi, K., Denniston, C., Ginting, M. Fadhil, Rosinol, A., Reinke, A., Palieri, M., Shi, J., A, Chatterjee, Morrell, B., Agha-mohammadi, A., and Carlone, L. 2022. LAMP 2.0: A Robust Multi-Robot SLAM System for

- Operation in Challenging Large-Scale Underground Environments. *IEEE Robotics and Automation Letters*, **7**(4), 9175–9182. .
- [166] Chang, Y., Ballotta, L., and Carlone, L. 2023b. D-Lite: Navigation-Oriented Compression of 3D Scene Graphs under Communication Constraints. *IEEE Robotics and Automation Letters*.
- [167] Chang, Y., Hughes, N., Ray, A., and Carlone, L. 2023c. Hydra-Multi: Collaborative Online Construction of 3D Scene Graphs with Multi-Robot Teams. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [168] Chang, Y., Fermoselle, L., Ta, D., Bucher, B., Carlone, L., and Wang, J. 2025. ASHiTA: Automatic Scene-grounded Hierarchical Task Analysis. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [169] Charatan, David, Li, Sizhe, Tagliasacchi, Andrea, and Sitzmann, Vincent. 2024. pixelSplat: 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [170] Charrow, Benjamin, Kahn, Gregory, Patil, Sachin, Liu, Sikang, Goldberg, Ken, Abbeel, Pieter, Michael, Nathan, and Kumar, Vijay. 2015. Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping. In: *Robotics: Science and Systems (RSS)*.
- [171] Chatfield, Averil B. 1997. *Fundamentals of High Accuracy Inertial Navigation*. AIAA.
- [172] Chatila, Raja, and Laumond, Jean-Paul. 1985. Position referencing and consistent world modeling for mobile robots. Pages 138–145 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [173] Chatterjee, A., and Govindu, V. M. 2013. Efficient and Robust Large-Scale Rotation Averaging. Pages 521–528 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [174] Chebrolu, Nived, Läbe, Thomas, Vysotska, Olga, Behley, Jens, and Stachniss, Cyrill. 2020. Adaptive Robust Kernels for Non-Linear Least Squares Problems. *arXiv preprint*.
- [175] Chebrolu, Nived, Magistri, Federico, Läbe, Thomas, and Stachniss, Cyrill. 2021. Registration of spatio-temporal point clouds of plants for phenotyping. *PLOS ONE*, **16**(2), e0247243.
- [176] Checchin, Paul, Gérossier, Franck, Blanc, Christophe, Chapuis, Roland, and Trassoudaine, Laurent. 2010. Radar Scan Matching SLAM Using the Fourier-Mellin Transform. Pages 151–161 of: Howard, Andrew, Iagnemma, Karl, and Kelly, Alonzo (eds), *Field and Service Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg.
- [177] Chen, B., Xu, Z., Kirmani, S., Ichter, B., Sadigh, D., Guibas, L., and Xia, F. 2024a. SpatialVLM: Endowing Vision-Language Models with Spatial Reasoning Capabilities. Pages 14455–14465 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [178] Chen, Boyuan, Xia, Fei, Ichter, Brian, Rao, Kanishka, Gopalakrishnan, Keerthana, Ryoo, Michael S, Stone, Austin, and Kappler, Daniel. 2023a. Open-vocabulary queryable scene representations for real world planning. Pages 11509–11522 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [179] Chen, Changhao, Lu, Xiaoxuan, Markham, Andrew, and Trigoni, Niki. 2018.

- Ionet: Learning to cure the curse of drift in inertial odometry. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32.
- [180] Chen, Chuchu, Geneva, Patrick, Peng, Yuxiang, Lee, Woosik, and Huang, Guoquan. 2023b. Optimization-Based VINS: Consistency, Marginalization, and FEJ. Pages 1517–1524 of: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- [181] Chen, Chuchu, Peng, Yuxiang, and Huang, Guoquan. 2024b (May). Fast and Consistent Covariance Recovery for Sliding-window Optimization-based VINS. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [182] Chen, Hansheng, Wang, Pichao, Wang, Fan, Tian, Wei, Xiong, Lu, and Li, Hao. 2022a. EPro-PnP: Generalized End-to-End Probabilistic Perspective-n-Points for Monocular Object Pose Estimation. Pages 2781–2790 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [183] Chen, Peiyu, Guan, Weipeng, and Lu, Peng. 2023c. ESVIO: Event-based Stereo Visual Inertial Odometry. *IEEE Robotics and Automation Letters*, **8**(6), 3661–3668.
- [184] Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyan, and Zhang, Zheng. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint*.
- [185] Chen, Ting, Kornblith, Simon, Norouzi, Mohammad, and Hinton, Geoffrey. 2020. A Simple Framework for Contrastive Learning of Visual Representations. Pages 1597–1607 of: *Intl. Conf. on Machine Learning (ICML)*, vol. 119. PMLR.
- [186] Chen, Xiangyu, Yang, Fan, and Wang, Chen. 2024c. iA\*: Imperative Learning-based A\* Search for Pathfinding. *arXiv preprint*.
- [187] Chen, Xieyuanli, Milioto, Andres, Palazzolo, Emanuele, Giguere, Philippe, Behley, Jens, and Stachniss, Cyrill. 2019. Suma++: Efficient lidar-based semantic slam. Pages 4530–4537 of: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [188] Chen, Xieyuanli, Läbe, Thomas, Milioto, Andres, Röhling, Timo, Behley, Jens, and Stachniss, Cyrill. 2022b. OverlapNet: A siamese network for computing LiDAR scan similarity with applications to loop closing and localization. *Autonomous Robots*, 1–21.
- [189] Chen, Y., Davis, T.A., Hager, W.W., and Rajamanickam, S. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/-Downdate. *ACM Trans. Math. Softw.*, **35**(3), 22:1–22:14.
- [190] Chen, Yongbo, Huang, Shoudong, Zhao, Liang, and Dissanayake, Gamini. 2021. Cramér–Rao bounds and optimal design metrics for pose-graph SLAM. *IEEE Trans. Robotics*, **37**(2), 627–641.
- [191] Chen, Yuedong, Xu, Haofei, Zheng, Chuanxia, Zhuang, Bohan, Pollefeys, Marc, Geiger, Andreas, Cham, Tat-Jen, and Cai, Jianfei. 2024d. MVSplat: Efficient 3D Gaussian Splatting from Sparse Multi-View Images. *European Conf. on Computer Vision (ECCV)*.
- [192] Chen, Zhiqin, and Zhang, Hao. 2019. Learning implicit fields for generative shape modeling. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [193] Cheng, Qing, Zeller, Niclas, and Cremers, Daniel. 2022a. Vision-based large-scale 3d semantic mapping for autonomous driving applications. Pages 9235–9242 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [194] Cheng, Tianheng, Song, Lin, Ge, Yixiao, Liu, Wenyu, Wang, Xinggang, and Shan, Ying. 2024. YOLO-World: Real-Time Open-Vocabulary Object Detection. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*.
- [195] Cheng, Yuwei, Su, Jingran, Jiang, Mengxin, and Liu, Yimin. 2022b. A Novel Radar Point Cloud Generation Method for Robot Environment Perception. *IEEE Trans. Robotics*.
- [196] Chhatkuli, Ajad, Pizarro, Daniel, and Bartoli, Adrien. 2014. Non-Rigid Shape-from-Motion for Isometric Surfaces using Infinitesimal Planarity. In: *British Machine Vision Conf. (BMVC)*.
- [197] Chhatkuli, Ajad, Pizarro, Daniel, Collins, Toby, and Bartoli, Adrien. 2016. Inextensible Non-Rigid Shape-from-Motion by second-order cone programming. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [198] Chhatkuli, Ajad, Pizarro, Daniel, Bartoli, Adrien, and Collins, Toby. 2017. A stable analytical framework for isometric shape-from-template by surface integration. *IEEE Trans. Pattern Anal. Machine Intell.*, **39**(5), 833–850.
- [199] Chi, Cheng, Xu, Zhenjia, Feng, Siyuan, Cousineau, Eric, Du, Yilun, Burchfiel, Benjamin, Tedrake, Russ, and Song, Shuran. 2024a. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *Intl. J. of Robotics Research*.
- [200] Chi, Cheng, Xu, Zhenjia, Pan, Chuer, Cousineau, Eric, Burchfiel, Benjamin, Feng, Siyuan, Tedrake, Russ, and Song, Shuran. 2024b. Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots. In: *Robotics: Science and Systems (RSS)*.
- [201] Chiang, Hao-Tien Lewis, Xu, Zhuo, Fu, Zipeng, Jacob, Mithun George, Zhang, Tingnan, Lee, Tsang-Wei Edward, Yu, Wenhao, Schenck, Connor, Rendleman, David, Shah, Dhruv, et al. 2024. Mobility vla: Multimodal instruction navigation with long-context vlms and topological graphs. *arXiv preprint*.
- [202] Chilian, Annett, Hirschmüller, Heiko, and Görner, Martin. 2011. Multisensor data fusion for robust pose estimation of a six-legged walking robot. Pages 2497–2504 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [203] Chin, T., Kee, Y. H., Eriksson, A., and Neumann, F. 2016 (June). Guaranteed Outlier Removal with Mixed Integer Linear Programs. Pages 5858–5866 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [204] Chin, Tat-Jun, Bagchi, Samya, Eriksson, Anders P., and van Schaik, André. 2019. Star Tracking using an Event Camera. Pages 1646–1655 of: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops*.
- [205] Chirikjian, G. S. 2009. *Stochastic Models, Information Theory, and Lie Groups, Volume 1: Classical Results and Geometric Methods (Applied and Numerical Harmonic Analysis)*. Birkhauser.
- [206] Chirikjian, G. S. 2012. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications (Applied and Numerical Harmonic Analysis)*. Birkhauser.

- [207] Chirkjian, G. S., and Kyatkin, A. B. 2001. *Engineering Applications of Non-commutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups*. Boca Raton: CRC Press.
- [208] Chitta, Sachin, Vemaza, Paul, Geykhman, Roman, and Lee, Daniel D. 2007. Proprioceptive localization for a quadrupedal robot on known terrain. Pages 4582–4587 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [209] Chiuso, A., Favaro, P., Jin, Hailin, and Soatto, S. 2002. Structure from motion causally integrated over time. *IEEE Trans. Pattern Anal. Machine Intell.*, **24**(4), 523–535.
- [210] Cho, Younggun, Kim, Giseop, and Kim, Ayoung. 2020. Unsupervised Geometry-Aware Deep LiDAR Odometry. Pages 2145–2152 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [211] Cho, Younghun, Kim, Giseop, Lee, Sangmin, and Ryu, Jee-Hwan. 2022. Openstreetmap-based lidar global localization in urban environment without a prior lidar map. *IEEE Robotics and Automation Letters*, **7**(2), 4999–5006.
- [212] Choi, Minseong, Yang, Seunghoon, Han, Seungho, Lee, Yeongseok, Lee, Minyoung, Choi, Keun Ha, and Kim, Kyung-Soo. 2023. MSC-RAD4R: ROS-Based Automotive Dataset With 4D Radar. *IEEE Robotics and Automation Letters*, **8**(11), 7194–7201.
- [213] Choset, Howie, and Nagatani, Keiji. 2001. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Trans. Robot. Automat.*, **17**(2), 125–137.
- [214] Choy, Christopher B, Gwak, JunYoung, Savarese, Silvio, and Chandraker, Manmohan. 2016. Universal correspondence network. *Advances in neural information processing systems*, **29**.
- [215] Chum, O., and Matas, J. 2005. Matching with PROSAC - Progressive Sample Consensus. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [216] Chum, Ondřej, Matas, Jiří, and Kittler, Josef. 2003. Locally optimized RANSAC. Pages 236–243 of: *Joint Pattern Recognition Symposium*. Springer.
- [217] Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint*.
- [218] Chung, Timothy H., Orekhov, Viktor, and Maio, Angela. 2023. Into the Robotic Depths: Analysis and Insights from the DARPA Subterranean Challenge. *Annual Review of Control, Robotics, and Autonomous Systems*, **6**(1).
- [219] Churchill, Winston, and Newman, Paul. 2013. Experience-based navigation for long-term localisation. *Intl. J. of Robotics Research*, **32**(14), 1645–1661.
- [220] Cioffi, Giovanni, Bauersfeld, Leonard, Kaufmann, Elia, and Scaramuzza, Davide. 2023. Learned inertial odometry for autonomous drone racing. *IEEE Robotics and Automation Letters*, **8**(5), 2684–2691.
- [221] Cipolla, Roberto, Gal, Yarin, and Kendall, Alex. 2018. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. Pages 7482–7491 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [222] Civera, Javier, Grasa, Oscar G., Davison, Andrew J., and Montiel, J. M. M.

2010. 1-Point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry. *J. of Field Robotics*, **27**(5), 609–631.
- [223] Clark, J.H. 1976. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, **19**(1), 547–554.
- [224] Clark, Steve, and Durrant-Whyte, Hugh. 1998. Autonomous land vehicle navigation using millimeter wave radar. Pages 3697–3702 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4. IEEE.
- [225] Cohen, Nadav, and Klein, Itzik. 2024. *Inertial Navigation Meets Deep Learning: A Survey of Current Trends and Future Directions*.
- [226] Collaboration, Open X-Embodiment, O'Neill, Abby, Rehman, Abdul, Gupta, Abhinav, Maddukuri, Abhiram, Gupta, Abhishek, Padalkar, Abhishek, Lee, Abraham, Pooley, Acorn, Gupta, Agrim, Mandlekar, Ajay, Jain, Ajinkya, Tung, Albert, Bewley, Alex, Herzog, Alex, Irpan, Alex, Khazatsky, Alexander, Rai, Anant, Gupta, Anchit, Wang, Andrew, Kolobov, Andrey, Singh, Anikait, Garg, Animesh, Kembhavi, Aniruddha, Xie, Annie, Brohan, Anthony, Raffin, Antonin, Sharma, Archit, Yavary, Arefeh, Jain, Arhan, Balakrishna, Ashwin, Wahid, Ayzaan, Burgess-Limerick, Ben, Kim, Beomjoon, Schölkopf, Bernhard, Wulfe, Blake, Ichter, Brian, Lu, Cewu, Xu, Charles, Le, Charlotte, Finn, Chelsea, Wang, Chen, Xu, Chenfeng, Chi, Cheng, Huang, Chenguang, Chan, Christine, Agia, Christopher, Pan, Chuer, Fu, Chuyuan, Devin, Coline, Xu, Danfei, Morton, Daniel, Driess, Danny, Chen, Daphne, Pathak, Deepak, Shah, Dhruv, Büchler, Dieter, Jayaraman, Dinesh, Kalashnikov, Dmitry, Sadigh, Dorsa, Johns, Edward, Foster, Ethan, Liu, Fangchen, Ceola, Federico, Xia, Fei, Zhao, Feiyu, Frujeri, Felipe Vieira, Stulp, Freek, Zhou, Gaoyue, Sukhatme, Gaurav S., Salhotra, Gautam, Yan, Ge, Feng, Gilbert, Schiavi, Giulio, Berseth, Glen, Kahn, Gregory, Yang, Guangwen, Wang, Guanzhi, Su, Hao, Fang, Hao-Shu, Shi, Haochen, Bao, Henghui, Amor, Heni Ben, Christensen, Henrik I., Furuta, Hiroki, Bharadhwaj, Homanga, Walke, Homer, Fang, Hongjie, Ha, Huy, Mordatch, Igor, Radosavovic, Ilija, Leal, Isabel, Liang, Jacky, Abou-Chakra, Jad, Kim, Jaehyun, Drake, Jaimyn, Peters, Jan, Schneider, Jan, Hsu, Jasmine, Vakil, Jay, Bohg, Jeannette, Bingham, Jeffrey, Wu, Jeffrey, Gao, Jensen, Hu, Jiaheng, Wu, Jiajun, Wu, Jialin, Sun, Jiankai, Luo, Jianlan, Gu, Jiayuan, Tan, Jie, Oh, Jihoon, Wu, Jimmy, Lu, Jingpei, Yang, Jingyun, Malik, Jitendra, Silvério, João, Hejna, Joey, Booher, Jonathan, Tompson, Jonathan, Yang, Jonathan, Salvador, Jordi, Lim, Joseph J., Han, Junhyek, Wang, Kaiyuan, Rao, Kanishka, Pertsch, Karl, Hausman, Karol, Go, Keegan, Gopalakrishnan, Keerthana, Goldberg, Ken, Byrne, Kendra, Oslund, Kenneth, Kawaharazuka, Kento, Black, Kevin, Lin, Kevin, Zhang, Kevin, Ehsani, Kiana, Lekkala, Kiran, Ellis, Kirsty, Rana, Krishan, Srinivasan, Krishnan, Fang, Kuan, Singh, Kunal Pratap, Zeng, Kuo-Hao, Hatch, Kyle, Hsu, Kyle, Itti, Laurent, Chen, Lawrence Yunliang, Pinto, Lerrel, Fei-Fei, Li, Tan, Liam, Fan, Linxi "Jim", Ott, Lionel, Lee, Lisa, Weihs, Luca, Chen, Magnum, Lepert, Marion, Memmel, Marius, Tomizuka, Masayoshi, Itkina, Masha, Castro, Mateo Guaman, Spero, Max, Du, Maximilian, Ahn, Michael, Yip, Michael C., Zhang, Mingtong, Ding, Mingyu, Heo, Minho, Srirama, Mohan Kumar, Sharma, Mohit, Kim, Moo Jin, Kanazawa, Naoaki, Hansen, Nicklas, Heess, Nicolas, Joshi, Nikhil J., Suenderhauf, Niko, Liu, Ning,

- Palo, Norman Di, Shafiullah, Nur Muhammad Mahi, Mees, Oier, Kroemer, Oliver, Bastani, Osbert, Sanketi, Pannag R, Miller, Patrick "Tree", Yin, Patrick, Wohlhart, Paul, Xu, Peng, Fagan, Peter David, Mitrano, Peter, Sermanet, Pierre, Abbeel, Pieter, Sundaresan, Priya, Chen, Qiuyu, Vuong, Quan, Rafailov, Rafael, Tian, Ran, Doshi, Ria, Mart'in-Mart'in, Roberto, Baijal, Rohan, Scalise, Rosario, Hendrix, Rose, Lin, Roy, Qian, Runjia, Zhang, Ruohan, Mendonca, Russell, Shah, Rutav, Hoque, Ryan, Julian, Ryan, Bustamante, Samuel, Kirmani, Sean, Levine, Sergey, Lin, Shan, Moore, Sherry, Bahl, Shikhar, Dass, Shivin, Sonawani, Shubham, Tulsiani, Shubham, Song, Shuran, Xu, Sichun, Haldar, Siddhant, Karamcheti, Siddharth, Adebola, Simeon, Guist, Simon, Nasiriany, Soroush, Schaal, Stefan, Welker, Stefan, Tian, Stephen, Ramamoorthy, Subramanian, Dasari, Sudeep, Belkhale, Suneel, Park, Sungjae, Nair, Suraj, Mirchandani, Suvir, Osa, Takayuki, Gupta, Tanmay, Harada, Tatsuya, Matsushima, Tatsuya, Xiao, Ted, Kollar, Thomas, Yu, Tianhe, Ding, Tianli, Davchev, Todor, Zhao, Tony Z., Armstrong, Travis, Darrell, Trevor, Chung, Trinity, Jain, Vidhi, Kumar, Vikash, Vanhoucke, Vincent, Zhan, Wei, Zhou, Wenxuan, Burgard, Wolfram, Chen, Xi, Chen, Xiangyu, Wang, Xiaolong, Zhu, Xing-hao, Geng, Xinyang, Liu, Xiyuan, Liangwei, Xu, Li, Xuanlin, Pang, Yang-song, Lu, Yao, Ma, Yecheng Jason, Kim, Yejin, Chebotar, Yevgen, Zhou, Yifan, Zhu, Yifeng, Wu, Yilin, Xu, Ying, Wang, Yixuan, Bisk, Yonatan, Dou, Yongqiang, Cho, Yoonyoung, Lee, Youngwoon, Cui, Yuchen, Cao, Yue, Wu, Yueh-Hua, Tang, Yujin, Zhu, Yuke, Zhang, Yunchu, Jiang, Yunfan, Li, Yunshuang, Li, Yunzhu, Iwasawa, Yusuke, Matsuo, Yutaka, Ma, Zehan, Xu, Zhuo, Cui, Zichen Jeff, Zhang, Zichen, Fu, Zipeng, and Lin, Zipeng. 2023. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*.
- [227] Collins, T., and Bartoli, A. 2010. Locally affine and planar deformable surface reconstruction from video. In: *International Workshop on Vision, Modeling and Visualization*.
  - [228] Collobert, Ronan, Bengio, Samy, and Mariéthoz, Johnny. 2002. *Torch: a modular machine learning software library*. Tech. rept. Idiap.
  - [229] Cook, Matthew, Gugelmann, Luca, Jug, Florian, Krautz, Christoph, and Steger, Angelika. 2011. Interacting maps for fast visual interpretation. Pages 770–776 of: *Int. Joint Conf. Neural Netw. (IJCNN)*.
  - [230] Cooper, Gregory F. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, **42**(2-3), 393–405.
  - [231] Cootes, Timothy F., Taylor, Christopher J., Cooper, David H., and Graham, Jim. 1995. Active shape models - their training and application. *Comput. Vis. Image Underst.*, **61**(1), 38–59.
  - [232] Crassidis, J.L. 2006. Sigma-point Kalman filtering for integrated GPS and inertial navigation. *IEEE Trans. Aerosp. Electron. Syst.*, **42**(2), 750–756.
  - [233] Crowley, J.L. 1989. World modeling and position estimation for a mobile robot using ultra-sonic ranging. Pages 674–680 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [234] Curless, Brian, and Levoy, Marc. 1996. A volumetric method for building complex models from range images. Pages 303–312 of: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
  - [235] Czarnowski, J., Leutenegger, S., and Davison, A. J. 2017. Semantic Texture

- for Robust Dense Tracking. In: *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*.
- [236] Czarnowski, Jan, Laidlow, Tristan, Clark, Ronald, and Davison, Andrew J. 2020. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, **5**(2), 721–728.
  - [237] Dai, Angela, Nießner, Matthias, Zollhöfer, Michael, Izadi, Shahram, and Theobalt, Christian. 2017a. BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration. *ACM Trans. on Graphics (TOG)*, **36**(3).
  - [238] Dai, Angela, Chang, Angel X, Savva, Manolis, Halber, Maciej, Funkhouser, Thomas, and Nießner, Matthias. 2017b. Scannet: Richly-annotated 3d reconstructions of indoor scenes. Pages 5828–5839 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
  - [239] Dai, Yuchao, Li, Hongdong, and He, Mingyi. 2014. A simple prior-free method for non-rigid structure-from-motion factorization. *Intl. J. of Computer Vision*, **107**(2), 101–122.
  - [240] Davis, T.A. 2011. Algorithm 915: SuiteSparseQR, A multifrontal multi-threaded sparse QR factorization package. *ACM Trans. Math. Softw.*, **38**(1), 8:1–8:22.
  - [241] Davis, T.A., Gilbert, J.R., Larimore, S.I., and Ng, E.G. 2004. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, **30**(3), 353–376.
  - [242] Davison, A. J., and Ortiz, J. 2019. FutureMapping 2: Gaussian Belief Propagation for Spatial AI. *arXiv preprint*.
  - [243] Davison, A. J., Molton, N. D., Reid, I., and Stasse, O. 2007. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Trans. Pattern Anal. Machine Intell.*, **29**(6), 1052–1067.
  - [244] Davison, Andrew J. 2003 (Oct). Real-time simultaneous localisation and mapping with a single camera. Pages 1403–1410, vol. 2 of: *Intl. Conf. on Computer Vision (ICCV)*.
  - [245] Davison, Andrew J. 2018. FutureMapping: The computational structure of spatial AI systems. *arXiv preprint*.
  - [246] De Martini, Daniele, Gadd, Matthew, and Newman, Paul. 2020. kRadar++: Coarse-to-Fine FMCW Scanning Radar Localisation. *Sensors*, **20**(21).
  - [247] Dechter, Rina, and Mateescu, Robert. 2007. AND/OR Search Spaces for Graphical Models. *Artificial Intelligence*, **171**(2-3), 73–106.
  - [248] Deems, Jeffrey S., Painter, Thomas H., and Finnegan, David C. 2013. Lidar measurement of snow depth: a review. *Journal of Glaciology*, **59**(215), 467–479.
  - [249] Dellaert, F. 2005. Square Root SAM: Simultaneous Location and Mapping via Square Root Information Smoothing. In: *Robotics: Science and Systems (RSS)*.
  - [250] Dellaert, F. 2021. Factor Graphs: Exploiting Structure in Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, **4**, 141–166.
  - [251] Dellaert, F., Rosen, D.M., Wu, J., Mahony, R., and Carlone, L. 2020. Shonan Rotation Averaging: Global Optimality by Surfing  $SO(p)^n$ . In: *European Conf. on Computer Vision (ECCV)*.
  - [252] Dellaert, Frank. 2012. *Factor graphs and GTSAM: A hands-on introduction*. Tech. rept. Georgia Institute of Technology.

- [253] Dellaert, Frank, and Contributors, GTSAM. 2022 (May). *GTSAM: Georgia Tech Smoothing and Mapping Library*. Georgia Tech Borg Lab.
- [254] Dellaert, Frank, and Kaess, Michael. 2006. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, **25**(12), 1181–1203.
- [255] Dellaert, Frank, Kaess, Michael, et al. 2017. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, **6**(1-2), 1–139.
- [256] Dellenbach, Pierre, Deschaud, Jean-Emmanuel, Jacquet, Bastien, and Goulette, François. 2022. CT-ICP: Real-Time Elastic LiDAR Odometry with Loop Closure. Pages 5580–5586 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [257] Delmerico, Jeffrey, Cieslewski, Titus, Rebecq, Henri, Faessler, Matthias, and Scaramuzza, Davide. 2019. Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset. Pages 6713–6719 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [258] Demmel, N, Sommer, C, Cremers, D, and Usenko, V. 2021a. Square Root Bundle Adjustment for Large-Scale Reconstruction. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [259] Demmel, N, Schubert, D, Sommer, C, Cremers, D, and Usenko, V. 2021b. Square Root Marginalization for Sliding-Window Bundle Adjustment. In: *IEEE International Conference on Computer Vision (ICCV)*.
- [260] Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. 2009. Imagenet: A large-scale hierarchical image database. Pages 248–255 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. Ieee.
- [261] Deng, Junyuan, Chen, Xieyuanli, Xia, Songpengcheng, Sun, Zhen, Liu, Guoqing, Yu, Wenxian, and Pei, Ling. 2023a. NeRF-LOAM: Neural Implicit Representation for Large-Scale Incremental LiDAR Odometry and Mapping. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [262] Deng, Tianchen, Xie, Hongle, Wang, Jingchuan, and Chen, Weidong. 2023b. Long-term visual simultaneous localization and mapping: Using a bayesian persistence filter-based global map prediction. *IEEE Robotics and Automation Magazine*, **30**(1), 36–49.
- [263] Deng, Yinan, Wang, Jiahui, Zhao, Jingyu, Tian, Xinyu, Chen, Guangyan, Yang, Yi, and Yue, Yufeng. 2024. OpenGraph: Open-Vocabulary Hierarchical 3D Graph Representation in Large-Scale Outdoor Environments. *IEEE Robotics and Automation Letters*, **9**(10), 8402–8409.
- [264] Dennis, J.E., and Schnabel, R.B. 1983. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall.
- [265] Deschaud, J.-E. 2018. IMLS-SLAM: Scan-to-Model Matching Based on 3D Data. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [266] Deschênes, S.-P., Baril, D., Boxan, M., Laconte, J., Giguère, P., and Pomerleau, F. 2024. Saturation-Aware Angular Velocity Estimation: Extending the Robustness of SLAM to Aggressive Motions. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [267] DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. 2018. Superpoint: Self-supervised interest point detection and description. Pages 224–236 of: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops*.

- [268] Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. Pages 4171–4186 of: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*.
- [269] Dexheimer, E., and Davison, A. J. 2024. COMO: Compact Mapping and Odometry. In: *European Conf. on Computer Vision (ECCV)*.
- [270] Diebel, James. 2006. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Tech. rept. Stanford University.
- [271] Dissanayake, MWM Gamini, Newman, Paul, Clark, Steve, Durrant-Whyte, Hugh F, and Csorba, Michael. 2001. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on robotics and automation*, **17**(3), 229–241.
- [272] Doer, Christopher, and Trommer, Gert F. 2020. An EKF Based Approach to Radar Inertial Odometry. Pages 152–159 of: *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*.
- [273] Doer, Christopher, and Trommer, Gert F. 2021. Radar Visual Inertial Odometry and Radar Thermal Inertial Odometry: Robust Navigation even in Challenging Visual Conditions. Pages 331–338 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [274] Doherty, Kevin, Fourie, Dehann, and Leonard, John J. 2019. Multimodal Semantic SLAM with Probabilistic Data Association. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [275] Doherty, Kevin, Papalia, Alan, Huang, Yewei, Rosen, David, Englot, Brendan, and Leonard, John. 2024. MAC: Graph Sparsification by Maximizing Algebraic Connectivity. *arXiv preprint*.
- [276] Doherty, Kevin J, Lu, Ziqi, Singh, Kurran, and Leonard, John J. 2022. Discrete-Continuous Smoothing and Mapping. *IEEE Robotics and Automation Letters*, October.
- [277] Dong, Hang, and Barfoot, Timothy D. 2013. Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. Pages 327–342 of: *Field and Service Robotics (FSR)*. Springer.
- [278] Dosovitskiy, Alexey, Fischer, Philipp, Ilg, Eddy, Hausser, Philip, Hazirbas, Caner, Golkov, Vladimir, Van Der Smagt, Patrick, Cremers, Daniel, and Brox, Thomas. 2015. Flownet: Learning optical flow with convolutional networks. Pages 2758–2766 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [279] Dosovitskiy, Alexey, Ros, German, Codevilla, Felipe, Lopez, Antonio, and Koltun, Vladlen. 2017. CARLA: An Open Urban Driving Simulator. In: *Conf. on Robot Learning (CoRL)*.
- [280] Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, Uszkoreit, Jakob, and Houlsby, Neil. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *Intl. Conf. on Learning Representations (ICLR)*.
- [281] Driess, Danny, Xia, Fei, Sajjadi, Mehdi S. M., Lynch, Corey, Chowdhery, Aakanksha, Ichter, Brian, Wahid, Ayzaan, Tompson, Jonathan, Vuong, Quan, Yu, Tianhe, Huang, Wenlong, Chebotar, Yevgen, Sermanet, Pierre, Duckworth, Daniel, Levine, Sergey, Vanhoucke, Vincent, Hausman, Karol,

- Toussaint, Marc, Greff, Klaus, Zeng, Andy, Mordatch, Igor, and Florence, Pete. 2023. PaLM-E: An Embodied Multimodal Language Model. In: *arXiv preprint*.
- [282] Droeschel, D., and Behnke, S. 2018. Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [283] Duane, C Brown. 1971. Close-range camera calibration. *Photogramm. Eng.*, **37**(8), 855–866.
- [284] Dubé, Renaud, Gawel, Abel, Sommer, Hannes, Nieto, Juan, Siegwart, Roland, and Cadena, Cesar. 2017. An online multi-robot SLAM system for 3D LiDARs. Pages 1004–1011 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [285] Dubé, Renaud, Cramariuc, Andrei, Dugas, Daniel, Sommer, Hannes, Dymczyk, Marcin, Nieto, Juan, Siegwart, Roland, and Cadena, Cesar. 2020. SegMap: Segment-based mapping and localization using data-driven descriptors. *Intl. J. of Robotics Research*, **39**(2-3), 339–355.
- [286] Duberg, D., and Jensfelt, P. 2020. UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown. *IEEE Robotics and Automation Letters*, **5**(4), 6411–6418.
- [287] Duckett, T., Marsland, S., and Shapiro, J. 2002. Fast, On-line Learning of Globally Consistent Maps. *Autonomous Robots*, **12**(3), 287–300.
- [288] Duff, I. S., and Reid, J. K. 1983. The Multifrontal Solution of Indefinite Sparse Symmetric Linear Systems. *ACM Trans. Math. Softw.*, **9**(3), 302–325.
- [289] Duisterhof, Bardienus, Zust, Lojze, Weinzaepfel, Philippe, Leroy, Vincent, Cabon, Yohann, and Revaud, Jérôme. 2025. MAST3R-SfM: a Fully-Integrated Solution for Unconstrained Structure-from-Motion. In: *Intl. Conf. on 3D Vision (3DV)*.
- [290] Dumbgen, F., Holmes, C., Agro, B., and Barfoot, T. 2024. Toward Globally Optimal State Estimation Using Automatically Tightened Semidefinite Relaxations. *IEEE Trans. Robotics*, **40**, 4338–4358.
- [291] Dunkley, O., Engel, J., Sturm, J., and Cremers, D. 2014. Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems Workshops*.
- [292] Dunning, Iain, Huchette, Joey, and Lubin, Miles. 2017. JuMP: A modeling language for mathematical optimization. *SIAM review*, **59**(2), 295–320.
- [293] Durrant-Whyte, H.F. 1988. Uncertain geometry in robotics. *IEEE Trans. Robot. Automat.*, **4**(1), 23–31.
- [294] Durrant-Whyte, H.F., Rye, D., and Nebot, E. 1996. Localisation of automatic guided vehicles. Pages 613–625 of: Giralt, G., and Hirzinger, G. (eds), *Intl. Symp. of Robotics Research (ISRR)*. Springer-Verlag.
- [295] Dusmanu, Mihai, Rocco, Ignacio, Pajdla, Tomás, Pollefeys, Marc, Sivic, Josef, Torii, Akihiko, and Sattler, Torsten. 2019. D2-Net: A Trainable CNN for Joint Description and Detection of Local Features. Pages 8092–8101 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [296] Dymczyk, Marcin, Lnen, Simon, Cieslewski, Titus, Bosse, Michael, Siegwart, Roland, and Furgale, Paul. 2015 (May). The gist of maps-summarizing experience for lifelong localization. Pages 2767–2773 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [297] Dziri, Nouha, Lu, Ximing, Sclar, Melanie, Li, Xiang Lorraine, Jian, Liwei, Lin, Bill Yuchen, West, Peter, Bhagavatula, Chandra, Bras, Ronan Le, Hwang, Jena D, et al. 2023. Faith and Fate: Limits of Transformers on Compositionality. *arXiv preprint*.
- [298] Ebadi, K., Chang, Y., Palieri, M., Stephens, A., Hatteland, A., Heiden, E., Thakur, A., Morrell, B., Carbone, L., and Aghamohammadi, A. 2020. LAMP: Large-Scale Autonomous Mapping and Positioning for Exploration of Perceptually-Degraded Subterranean Environments. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [299] Ebadi, Kamak, Bernreiter, Lukas, Biggie, Harel, Catt, Gavin, Chang, Yun, Chatterjee, Arghya, Denniston, Christopher E, Deschênes, Simon-Pierre, Harlow, Kyle, Khattak, Shehryar, et al. 2023. Present and future of slam in extreme environments: The darpa subt challenge. *IEEE Trans. Robotics*, **40**, 936–959.
- [300] Eckenhoff, Kevin, Geneva, Patrick, and Huang, Guoquan. 2019. Closed-form preintegration methods for graph-based visual–inertial navigation. *Intl. J. of Robotics Research*, **38**(5), 563–586.
- [301] Eckenhoff, Kevin, Geneva, Patrick, and Huang, Guoquan. 2021. MIMCVINS: A Versatile and Resilient Multi-IMU Multi-Camera Visual-Inertial Navigation System. *IEEE Trans. Robotics*, **37**(5), 1360–1380.
- [302] Eigen, David, Puhrsch, Christian, and Fergus, Rob. 2014. Depth map prediction from a single image using a multi-scale deep network. Pages 2366–2374 of: *Conf. Neural Information Processing Systems (NIPS)*.
- [303] El Moudni, Anass, Morbidi, Fabio, Kramm, Sébastien, and Boutteau, Rémi. 2023. An Event-based Stereo 3D Mapping and Tracking Pipeline for Autonomous Vehicles. Pages 5962–5968 of: *IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*.
- [304] Elfes, A. 1989. Using occupancy grids for mobile robot perception and navigation. *Computer*, **22**(6), 46–57.
- [305] Elseberg, J., Borrmann, D., and Nüchter, A. 2013. One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS J. of Photogrammetry and Remote Sensing (JPRS)*, **76**, 76–88.
- [306] Engel, J., Sturm, J., and Cremers, D. 2012 (Oct.). Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems Workshops*.
- [307] Engel, Jakob, Schöps, Thomas, and Cremers, Daniel. 2014. LSD-SLAM: Large-scale direct monocular SLAM. Pages 834–849 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [308] Engel, Jakob, Koltun, Vladlen, and Cremers, Daniel. 2017. Direct sparse odometry. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(3), 611–625.
- [309] Enqvist, O., Josephson, K., and Kahl, F. 2009. Optimal correspondences from pairwise constraints. Pages 1295–1302 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [310] Eriksson, A., Olsson, C., Kahl, F., and Chin, T.-J. 2018. Rotation averaging and strong duality. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [311] Ester, Martin, Kriegel, Hans-Peter, Sander, Jorg, Xu, Xiaowei, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases

- with noise. Pages 226–231 of: *Intl. Conf. Knowledge Discovery and Data Mining (KDD)*, vol. 96.
- [312] Eustice, R., Singh, H., and Leonard, J. 2005a (Apr.). Exactly Sparse Delayed-State Filters. Pages 2417–2424 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [313] Eustice, R., Walter, M., and Leonard, J. 2005b (Aug). Sparse Extended Information Filters: Insights into Sparsification. Pages 3281–3288 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
  - [314] Fahmi, Shamel, Fink, Geoff, and Semini, Claudio. 2021. On State Estimation for Legged Locomotion Over Soft Terrain. *IEEE Sensors Letters*, **5**(1), 1–4.
  - [315] Fallon, Maurice F., Antone, Matthew, Roy, Nicholas, and Teller, Seth. 2014. Drift-free humanoid state estimation fusing kinematic, inertial and LIDAR sensing. Pages 112–119 of: *IEEE Intl. Conf. on Humanoid Robots*.
  - [316] Fallon, Maurice F., Marion, Pat, Deits, Robin, Whelan, Thomas, Antone, Matthew E., McDonald, John, and Tedrake, Russ. 2015. Continuous humanoid locomotion over uneven terrain using stereo fusion. Pages 881–888 of: *IEEE Intl. Conf. on Humanoid Robots*.
  - [317] Faramarzi, Farnaz, Linares-Barranco, Bernabé, and Serrano-Gotarredona, Teresa. 2024. A  $128 \times 128$  Electronically Multi-Foveated Dynamic Vision Sensor With Real-Time Resolution Reconfiguration. *IEEE Access*, **12**, 192656–192671.
  - [318] Farrell, Jay A. 2008. *Aided Navigation: GPS with High Rate Sensors*. McGraw-Hill.
  - [319] Featherstone, Roy. 2007. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag.
  - [320] Feder, Tomás, and Vardi, Moshe Y. 1993. Monotone monadic SNP and constraint satisfaction. Pages 612–622 of: *ACM Symp. on Theory of Computing (STOC)*. New York, NY, USA: ACM Press.
  - [321] Fehr, Marius, Furrer, Fadri, Dryanovski, Ivan, Sturm, Jürgen, Gilitschenski, Igor, Siegwart, Roland, and Cadena, Cesar. 2017. TSDF-based change detection for consistent long-term dense reconstruction and dynamic object discovery. Pages 5237–5244 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [322] Feige, Uriel, Goldwasser, Shafi, Lovász, László, Safra, Shmuel, and Szegedy, Mario. 1991 (Sept.). Approximating clique is almost NP-complete. Pages 2–12 of: *Symp. on Foundations of Computer Science*.
  - [323] Feng, Qiaojun, Meng, Yue, Shan, Mo, and Atanasov, Nikolay. 2019. Localization and Mapping using Instance-specific Mesh Models. Pages 4985–4991 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
  - [324] Fent, Felix, Kuttnerreich, Fabian, Ruch, Florian, Rizwin, Farija, Juergens, Stefan, Lechermann, Lorenz, Nissler, Christian, Perl, Andrea, Voll, Ulrich, Yan, Min, and Lienkamp, Markus. 2024. MAN TruckScenes: A multimodal dataset for autonomous trucking in diverse conditions. In: *Advances in Neural Information Processing Systems (NIPS)*.
  - [325] Fernandez-Cortizas, M., Bavle, H., Perez-Saura, D., Sanchez-Lopez, J., Campoy, P., and Voos, H. 2024. Multi S-Graphs: An Efficient Distributed Semantic-Relational Collaborative SLAM. *IEEE Robotics and Automation Letters*, **9**(6), 6004–6011.

- [326] Finateu, Thomas, Niwa, Atsumi, Matolin, Daniel, Tsuchimoto, Koya, Mascheroni, Andrea, Reynaud, Etienne, Mostafalu, Pooria, Brady, Frederick, Chotard, Ludovic, LeGoff, Florian, Takahashi, Hirotsugu, Wakabayashi, Hayato, Oike, Yusuke, and Posch, Christoph. 2020. A 1280x720 Back-Illuminated Stacked Temporal Contrast Event-Based Vision Sensor with  $4.86\mu\text{m}$  Pixels, 1.066GEPS Readout, Programmable Event-Rate Controller and Compressive Data-Formatting Pipeline. Pages 112–114 of: *IEEE Int. Solid-State Circuits Conf. (ISSCC)*.
- [327] Finman, Ross, Whelan, Thomas, Kaess, Michael, and Leonard, John J. 2013. Toward lifelong object segmentation from change detection in dense rgbd maps. Pages 178–185 of: *European Conf. on Mobile Robots (ECMR)*. IEEE.
- [328] Fischler, Martin A, and Bolles, Robert C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24**(6), 381–395.
- [329] Focchi, Michele, Orsolino, Romeo, Camurri, Marco, Barasuol, Victor, Mastalli, Carlos, Caldwell, Darwin G., and Semini, Claudio. 2019. *Heuristic Planning for Rough Terrain Locomotion in Presence of External Disturbances and Variable Perception Quality*. Springer International Publishing. Pages 165–209.
- [330] Fong, W., Mohan, R., Hurtado, J., Zhou, L., Caesar, H., Beijbom, O., and Valada, A. 2022. Panoptic nuScenes: A Large-Scale Benchmark for LiDAR Panoptic Segmentation and Tracking. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [331] Fornasier, A., van Goor, P., Allak, E., Mahony, R., and Weiss, S. 2024. MSCEqF: A Multi State Constraint Equivariant Filter for Vision-Aided Inertial Navigation. *IEEE Robotics and Automation Letters*, **9**(1), 731–738.
- [332] Forsgren, Brendon, Kaess, Michael, Vasudevan, Ram, McLain, Timothy W., and Mangelson, Joshua G. 2024. Group-k consistent measurement set maximization via maximum clique over k-uniform hypergraphs for robust multi-robot map merging. *Intl. J. of Robotics Research*.
- [333] Forster, Christian, Pizzoli, Matia, and Scaramuzza, Davide. 2014. SVO: Fast semi-direct monocular visual odometry. Pages 15–22 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [334] Forster, Christian, Carbone, Luca, Dellaert, Frank, and Scaramuzza, Davide. 2015 (July 13–17.). IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In: *Robotics: Science and Systems (RSS)*.
- [335] Forster, Christian, Carbone, Luca, Dellaert, Frank, and Scaramuzza, Davide. 2016. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Trans. Robotics*, **33**(1), 1–21.
- [336] Fourie, Dehann. 2017. *Multi-modal and inertial sensor solutions for navigation-type factor graphs*. Ph.D. thesis, MIT.
- [337] Fourie, Dehann, Leonard, John, and Kaess, Michael. 2016. A nonparametric belief solution to the Bayes tree. Pages 2189–2196 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [338] Fredriksson, Johan, and Olsson, Carl. 2012. Simultaneous multiple rotation averaging using lagrangian duality. Pages 245–258 of: *Asian Conf. on Computer Vision (ACCV)*. Springer.

- [339] Frese, U. 2005. Treemap: An  $O(\log n)$  Algorithm for Simultaneous Localization and Mapping. Pages 455–476 of: *Spatial Cognition IV*. Springer Verlag.
- [340] Frese, U., Larsson, P., and Duckett, T. 2005. A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping. *IEEE Trans. Robotics*, **21**(2), 196–207.
- [341] Fridovich-Keil, Sara, Yu, Alex, Tancik, Matthew, Chen, Qinhong, Recht, Benjamin, and Kanazawa, Angjoo. 2022. Plenoxels: Radiance fields without neural networks. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [342] Fritzsche, Paul, Kueppers, Simon, Briese, Gunnar, and Wagner, Bernardo. 2016. Radar and LiDAR Sensorfusion in Low Visibility Environments. Pages 30–36 of: *Intl. Conf. on Informatics in Control (ICINCO)*.
- [343] Fritzsche, Paul, Kueppers, Simon, Briese, Gunnar, and Wagner, Bernardo. 2017. *Fusing LiDAR and Radar Data to Perform SLAM in Harsh Environments*. Springer International Publishing. Pages 175–189.
- [344] Fu, Jiahui, Huang, Qiangqiang, Doherty, Kevin, Wang, Yue, and Leonard, John J. 2021. A multi-hypothesis approach to pose ambiguity in object-based SLAM. Pages 7639–7646 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [345] Fu, Jiahui, Du, Yilun, Singh, Kurran, Tenenbaum, Joshua B, and Leonard, John J. 2023. Neuse: Neural se (3)-equivariant embedding for consistent spatial understanding with objects. *Robotics: Science and Systems (RSS)*.
- [346] Fu, Taimeng, Su, Shaoshu, Lu, Yiren, and Wang, Chen. 2024. iSLAM: Imperative SLAM. *IEEE Robotics and Automation Letters (RA-L)*.
- [347] Funk, N., Tarrio, J., Papatheodorou, S., Popović, M., Alcantarilla, P. F., and Leutenegger, S. 2021. Multi-resolution 3D Mapping with Explicit Free Space Representation for Fast and Accurate Mobile Robot Motion Planning. *IEEE Robotics and Automation Letters*, **6**(2), 3553–3560.
- [348] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. 2014. The SpiNNaker Project. *Proceedings of the IEEE*, **102**, 652–665.
- [349] Furgale, P., Barfoot, T.D., and Sibley, G. 2012. Continuous-time batch estimation using temporal basis functions. Pages 2088–2095 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [350] Furgale, Paul, Rehder, Joern, and Siegwart, Roland. 2013. Unified temporal and spatial calibration for multi-sensor systems. Pages 1280–1286 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [351] Furrer, Fadri, Novkovic, Tonci, Fehr, Marius, Gawel, Abel, Grinvald, Margarita, Sattler, Torsten, Siegwart, Roland, and Nieto, Juan. 2018. Incremental object database: Building 3d models from multiple partial observations. Pages 6835–6842 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [352] Gadd, Matthew, De Martini, Daniele, and Newman, Paul. 2021. Contrastive Learning for Unsupervised Radar Place Recognition. Pages 344–349 of: *Intl. Conf. on Advanced Robotics (ICAR)*.
- [353] Gadd, Matthew, De Martini, Daniele, Bartlett, Oliver, Murcutt, Paul, Towlson, Matt, Widojo, Matthew, Muşat, Valentina, Robinson, Luke, Panagiotaki, Efimia, Pramatarov, Georgi, et al. 2024. OORD: The Oxford Offroad Radar Dataset. *arXiv preprint*.

- [354] Galeote-Luque, Andres, Kubelka, Vladimír, Magnusson, Martin, Ruiz-Sarmiento, Jose-Raul, and Gonzalez-Jimenez, Javier. 2024. Doppler-only Single-scan 3D Vehicle Odometry. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [355] Galindo, Cipriano, Saffiotti, Alessandro, Coradeschi, Silvia, Buschka, Pär, Fernandez-Madrigal, Juan-Antonio, and González, Javier. 2005. Multi-Hierarchical Semantic Maps for Mobile Robotics. Pages 3492–3497 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [356] Gallego, Guillermo, and Scaramuzza, Davide. 2017. Accurate Angular Velocity Estimation with an Event Camera. *IEEE Robotics and Automation Letters*, **2**(2), 632–639.
- [357] Gallego, Guillermo, Mueggler, Elias, and Sturm, Peter. 2017. Translation of "Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung" by Erwin Kruppa (1913). *arXiv preprint*.
- [358] Gallego, Guillermo, Lund, Jon E. A., Mueggler, Elias, Rebecq, Henri, Delbrück, Tobi, and Scaramuzza, Davide. 2018a. Event-based, 6-DOF Camera Tracking from Photometric Depth Maps. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(10), 2402–2412.
- [359] Gallego, Guillermo, Rebecq, Henri, and Scaramuzza, Davide. 2018b. A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation. Pages 3867–3876 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [360] Gallego, Guillermo, Gehrig, Mathias, and Scaramuzza, Davide. 2019. Focus Is All You Need: Loss Functions For Event-based Vision. Pages 12272–12281 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [361] Gallego, Guillermo, Delbrück, Tobi, Orchard, Garrick, Bartolozzi, Chiara, Taba, Brian, Censi, Andrea, Leutenegger, Stefan, Davison, Andrew, Conradt, Jörg, Daniilidis, Kostas, and Scaramuzza, Davide. 2022. Event-based Vision: A Survey. *IEEE Trans. Pattern Anal. Machine Intell.*, **44**(1), 154–180.
- [362] Gálvez-López, Dorian, and Tardos, Juan D. 2012. Bags of binary words for fast place recognition in image sequences. *IEEE Trans. Robotics*, **28**(5), 1188–1197.
- [363] Gao, Ling, Liang, Yuxuan, Yang, Jiaqi, Wu, Shaoxun, Wang, Chenyu, Chen, Jiaben, and Kneip, Laurent. 2022. VECtor: A Versatile Event-Centric Benchmark for Multi-Sensor SLAM. *IEEE Robotics and Automation Letters*, **7**(3), 8217–8224.
- [364] Gao, Wei, and Tedrake, Russ. 2019. SurfelWarp: Efficient non-volumetric single view dynamic reconstruction. *arXiv preprint*.
- [365] Gao, X., Wang, R., Demmel, N., and Cremers, D. 2018 (October). LDSO: Direct Sparse Odometry with Loop Closure. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [366] Gao, Yiming, Cao, Yan-Pei, and Shan, Ying. 2023. SurfelNeRF: Neural Surfel Radiance Fields for Online Photorealistic Reconstruction of Indoor Scenes. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [367] Garcia-Salguero, Mercedes, Briales, Jesus, and Gonzalez-Jimenez, Javier. 2021. Certifiable relative pose estimation. *Image and Vision Computing*, **109**, 104142.

- [368] Garg, Ravi, Roussos, Anastasios, and Agapito, Lourdes. 2013. Dense variational reconstruction of non-rigid surfaces from monocular video. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [369] Gawel, Abel, Del Don, Carlo, Siegwart, Roland, Nieto, Juan, and Cadena, Cesar. 2018. X-view: Graph-based semantic multi-view localization. *IEEE Robotics and Automation Letters*, **3**(3), 1687–1694.
- [370] Gehrig, Daniel, Loquercio, Antonio, Derpanis, Konstantinos G., and Scaramuzza, Davide. 2019. End-to-End Learning of Representations for Asynchronous Event-Based Data. Pages 5632–5642 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [371] Gehrig, Daniel, Gehrig, Mathias, Hidalgo-Carrió, Javier, and Scaramuzza, Davide. 2020. Video to Events: Recycling Video Datasets for Event Cameras. Pages 3583–3592 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [372] Gehrig, Mathias, Aarents, Willem, Gehrig, Daniel, and Scaramuzza, Davide. 2021. DSEC: A Stereo Event Camera Dataset for Driving Scenarios. *IEEE Robotics and Automation Letters*, **6**(3), 4947–4954.
- [373] Geiger, A., Lenz, P., and Urtasun, R. 2012 (June). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. Pages 3354–3361 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [374] Geneva, Patrick, Eckenhoff, Kevin, Yang, Yulin, and Huang, Guoquan. 2018. Lips: Lidar-inertial 3d plane slam. Pages 123–130 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [375] Geneva, Patrick, Eckenhoff, Kevin, Lee, Woosik, Yang, Yulin, and Huang, Guoquan. 2020. OpenVINS: A Research Platform for Visual-Inertial Estimation. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [376] Gentil, Cedric Le, and Vidal-Calleja, Teresa. 2023. Continuous latent state preintegration for inertial-aided systems. *Intl. J. of Robotics Research*, **42**(10), 874–900.
- [377] Gentil, Cédric Le, Vayugundla, Mallikarjuna, Giubilato, Riccardo, Sturzl, Wolfgang, Vidal-Calleja, Teresa, and Triebel, Rudolph. 2020. Gaussian Process Gradient Maps for Loop-Closure Detection in Unstructured Planetary Environments. Pages 1895–1902 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [378] Ghaffari Jadidi, Maani, Valls Miro, Jaime, and Dissanayake, Gamini. 2018. Gaussian processes autonomous mapping and exploration for range-sensing mobile robots. *Autonomous Robots*, **42**(2), 273–290.
- [379] Ghiasi, Golnaz, Gu, Xiuye, Cui, Yin, and Lin, Tsung-Yi. 2022. Scaling open-vocabulary image segmentation with image-level labels. Pages 540–557 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [380] Ghosh, Suman, and Gallego, Guillermo. 2025. Event-based Stereo Depth Estimation: A Survey. *IEEE Trans. Pattern Anal. Machine Intell.*
- [381] Ghosh, Suman, Cavinato, Valentina, and Gallego, Guillermo. 2024. E-PTAM: Event-based Stereo Parallel Tracking and Mapping. Pages 70–87 of: *European Conf. on Computer Vision Workshops*.
- [382] Giamou, Matthew, Ma, Ziye, Peretroukhin, Valentin, and Kelly, Jonathan. 2019. Certifiably Globally Optimal Extrinsic Calibration From Per-Sensor Egomotion. *IEEE Robotics and Automation Letters*, **4**(2), 367–374.

- [383] Gifthaler, Markus, Neunert, Michael, Stäuble, Markus, and Buchli, Jonas. 2018. The control toolbox—an open-source c++ library for robotics, optimal and model predictive control. Pages 123–129 of: *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE.
- [384] Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. 2016. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Trans. Pattern Anal. Machine Intell.*, **38**(1), 142–158.
- [385] Gladkova, M, Wang, R, Zeller, N, and Cremers, D. 2021. Tight Integration of Feature-based Relocalization in Monocular Direct Visual Odometry. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [386] Gladkova, M, Korobov, N, Demmel, N, Ošep, A, Leal-Taixé, L, and Cremers, D. 2022. DirectTracker: 3D Multi-Object Tracking Using Direct Image Alignment and Photometric Bundle Adjustment. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [387] Godard, Clément, Mac Aodha, Oisin, and Brostow, Gabriel J. 2016. Unsupervised monocular depth estimation with left-right consistency. *arXiv preprint*.
- [388] Golkov, Vladimir, Skwark, Marcin J, Golkov, Antonij, Dosovitskiy, Alexey, Brox, Thomas, Meiler, Jens, and Cremers, Daniel. 2016. Protein contact prediction from amino acid co-evolution using convolutional networks for graph-valued images. *Advances in Neural Information Processing Systems (NIPS)*, **29**.
- [389] Golub, G.H., and Loan, C.F. Van. 1996. *Matrix Computations*. Third edn. Baltimore: Johns Hopkins University Press.
- [390] Gomez, Jorge, Patel, Saavan, Sarwar, Syed Shakib, Li, Ziyun, Capoccia, Rafaële, Wang, Zhao, Pinkham, Reid, Berkovich, Andrew, Tsai, Tsung-Hsun, De Salvo, Barbara, and Liu, Chiao. 2022. *Distributed On-Sensor Compute System for AR/VR Devices: A Semi-Analytical Simulation Framework for Power Estimation*.
- [391] Gómez-Rodríguez, Juan J, Lamarca, José, Morlana, Javier, Tardós, Juan D, and Montiel, J.M.M. 2021. SD-DefSLAM: Semi-direct monocular SLAM for deformable and intracorporeal scenes. Pages 5170–5177 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [392] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. 2016. *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [393] Gorlo, Nicolas, Schmid, Lukas, and Carlone, Luca. 2024. Long-Term Human Trajectory Prediction using 3D Dynamic Scene Graphs. *IEEE Robotics and Automation Letters*.
- [394] Gotardo, Paulo FU, and Martinez, Aleix M. 2011a. Kernel non-rigid structure from motion. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [395] Gotardo, Paulo FU, and Martinez, Aleix M. 2011b. Non-rigid structure from motion with complementary rank-3 spaces. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [396] Greve, Elias, Büchner, Martin, Vödisch, Niclas, Burgard, Wolfram, and Valada, Abhinav. 2024. Collaborative dynamic 3d scene graphs for automated driving. Pages 11118–11124 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.

- [397] Grimminger, Felix, Meduri, Avadesh, Khadiv, Majid, Viereck, Julian, Wüthrich, Manuel, Naveau, Maximilien, Berenz, Vincent, Heim, Steve, Widmaier, Felix, Flayols, Thomas, Fiene, Jonathan, Badri-Spröwitz, Alexander, and Righetti, Ludovic. 2020. An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research. *IEEE Robotics and Automation Letters*, **5**(2), 3650–3657.
- [398] Grinvald, Margarita, Furrer, Fadri, Novkovic, Tonci, Chung, Jen Jen, Cadenas, Cesar, Siegwart, Roland, and Nieto, Juan. 2019. Volumetric instance-aware semantic mapping and 3D object discovery. *IEEE Robotics and Automation Letters*, **4**(3), 3037–3044.
- [399] Grisetti, G., Stachniss, C., and Burgard, W. 2007. Improved Techniques for Grid Mapping With Rao-Blackwellized particle filters. *IEEE Trans. Robotics*, **23**(1), 34–46.
- [400] Grisetti, G., Stachniss, C., and Burgard, W. 2009. Non-linear Constraint Network Optimization for Efficient Map Learning. *Trans. on Intelligent Transportation systems*.
- [401] Grisetti, Giorgio, Kümmerle, Rainer, Stachniss, Cyrill, Frese, Udo, and Hertzberg, Christoph. 2010 (5). Hierarchical optimization on manifolds for online 2D and 3D mapping. Pages 273–278 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [402] Grisetti, Giorgio, Kümmerle, Rainer, Strasdat, Hauke, and Konolige, Kurt. 2011. g2o: A general framework for (hyper) graph optimization. Pages 9–13 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [403] Grohe, Martin, Neuen, Daniel, Schweitzer, Pascal, and Wiebking, Daniel. 2020. An Improved Isomorphism Test for Bounded-Tree-Width Graphs. *ACM Trans. on Algorithms (TALG)*, **16**(3).
- [404] Gropp, Amos, Yariv, Lior, Haim, Niv, Atzmon, Matan, and Lipman, Yaron. 2020. Implicit Geometric Regularization for Learning Shapes. Pages 3789–3799 of: *Intl. Conf. on Machine Learning (ICML)*.
- [405] Grupp, Michael. 2017. *evo: Python package for the evaluation of odometry and SLAM*. <https://github.com/MichaelGrupp/evo>.
- [406] Gu, Albert, Goel, Karan, and Ré, Christopher. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In: *The International Conference on Learning Representations (ICLR)*.
- [407] Gu, Qiao, Kuwajerwala, Ali, Morin, Sacha, Jatavallabhula, Krishna Murthy, Sen, Bipasha, Agarwal, Aditya, Rivera, Corban, Paul, William, Ellis, Kirsty, Chellappa, Rama, et al. 2024. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. Pages 5021–5028 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [408] Guan, Weipeng, Lin, Fuling, Chen, Peiyu, and Lu, Peng. 2024a. DEIO: Deep Event Inertial Odometry. *arXiv preprint*.
- [409] Guan, Weipeng, Chen, Peiyu, Zhao, Huibin, Wang, Yu, and Lu, Peng. 2024b. EVI-SAM: Robust, Real-Time, Tightly-Coupled Event–Visual–Inertial State Estimation and 3D Dense Mapping. *Adv. Intell. Syst.*, **6**(12), 2400243.
- [410] Guan, Weipeng, Chen, Peiyu, Xie, Yuhua, and Lu, Peng. 2024c. PL-EVIO: Robust Monocular Event-Based Visual Inertial Odometry With Point and Line Features. *IEEE Trans. Autom. Sci. Eng.*, **21**(4), 6277–6293.
- [411] Guédon, Antoine, and Lepetit, Vincent. 2024. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh

- rendering. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [412] Guennebaud, Gaël, Jacob, Benoit, et al. 2010. Eigen. URL: <http://eigen.tuxfamily.org>, **3**.
- [413] Guivant, J., and Nebot, E. 2001. Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Trans. Robot. Automat.*, **17**(3), 242–257.
- [414] Guo, Chao X., and Roumeliotis, Stergios I. 2013. IMU-RGBD camera navigation using point and plane features. Pages 3164–3171 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [415] Guo, Jiadong, Borges, Paulo VK, Park, Chanoh, and Gawel, Abel. 2019. Local descriptor for robust place recognition using lidar intensity. *IEEE Robotics and Automation Letters*, **4**(2), 1470–1477.
- [416] Guo, Shuang, and Gallego, Guillermo. 2024a. CMax-SLAM: Event-based Rotational-Motion Bundle Adjustment and SLAM System using Contrast Maximization. *IEEE Trans. Robotics*, **40**, 2442–2461.
- [417] Guo, Shuang, and Gallego, Guillermo. 2024b. Event-based Mosaicing Bundle Adjustment. Pages 479–496 of: *European Conf. on Computer Vision (ECCV)*.
- [418] Guo, Shuang, and Gallego, Guillermo. 2025. Event-based Photometric Bundle Adjustment. *IEEE Trans. Pattern Anal. Machine Intell.*
- [419] Guo, Yifan, Ren, Zhongqiang, and Wang, Chen. 2024. iMTSP: Solving Min-Max Multiple Traveling Salesman Problem with Imperative Learning. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [420] Gutmann, J.-S., and Konolige, K. 2000 (November). Incremental Mapping of Large Cyclic Environments. Pages 318–325 of: *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*.
- [421] Guzhov, Andrey, Raue, Federico, Hees, Jörn, and Dengel, Andreas. 2022. Audioclip: Extending clip to image, text and audio. Pages 976–980 of: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.
- [422] Gómez Rodríguez, Juan J., Montiel, J.M.M., and Tardós, Juan D. 2022. Tracking monocular camera pose and deformation for SLAM inside the human body. Pages 5278–5285 of: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- [423] Ha, D., and Schmidhuber, J. 2018. World Models. In: *Advances in Neural Information Processing Systems (NIPS)*.
- [424] Ha, Seongbo, Yeon, Jiung, and Yu, Hyeyoung. 2024. RGBD GS-ICP SLAM. In: *European Conf. on Computer Vision (ECCV)*.
- [425] Hadviger, Antea, Cvišić, Igor, Marković, Ivan, Vražić, Sacha, and Petrović, Ivan. 2021. Feature-based event stereo visual odometry. Pages 1–6 of: *European Conf. on Mobile Robots (ECMR)*.
- [426] Han, Luxin, Gao, Fei, Zhou, Boyu, and Shen, Shaojie. 2019. Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots. Pages 4423–4430 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [427] Hanan, S. 1989. Implementing Ray-tracing with Octrees and Neighbor Finding. *Computers & Graphics*, **13**(4), 445–460.

- [428] Handa, A., Newcombe, R. A., Angeli, A., and Davison, A. J. 2012. Real-Time Camera Tracking: When is High Frame-Rate Best? In: *European Conf. on Computer Vision (ECCV)*.
- [429] Handa, A., Whelan, T., McDonald, J. B., and Davison, A. J. 2014. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [430] Handa, Ankur, Bloesch, Michael, Pătrăucean, Viorica, Stent, Simon, McCormac, John, and Davison, Andrew. 2016. gynn: Neural network library for geometric computer vision. Pages 67–82 of: *European Conf. on Computer Vision Workshops*. Springer.
- [431] Harnad, Stevan. 1990. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, **42**(1), 335–346.
- [432] Harris, C., and Stephens, M. 1988. A combined corner and edge detector. Pages 147–151 of: *Proceedings of the Alvey Vision Conference*.
- [433] Harrison, Alastair, and Newman, Paul. 2008. High quality 3D laser ranging under general vehicle motion. Pages 7–12 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [434] Hart, William E, Laird, Carl D, Watson, Jean-Paul, Woodruff, David L, Hackebeil, Gabriel A, Nicholson, Bethany L, Siirola, John D, et al. 2017. *Pyomo-optimization modeling in python*. Vol. 67. Springer.
- [435] Hartley, R., Trumpf, J., Dai, Y., and Li, H. 2013. Rotation Averaging. *Intl. J. of Computer Vision*, **103**(3), 267–305.
- [436] Hartley, R.I., and Kahl, F. 2009. Global optimization through rotation space search. *Intl. J. of Computer Vision*, **82**(1), 64–79.
- [437] Hartley, Richard, and Zisserman, Andrew. 2003. *Multiple view geometry in computer vision*. Cambridge university press.
- [438] Hartley, Ross, Jadidi, Maani Ghaffari, Grizzle, Jessy, and Eustice, Ryan M. 2018a. Contact-Aided Invariant Extended Kalman Filtering for Legged Robot State Estimation. In: *Robotics: Science and Systems (RSS)*.
- [439] Hartley, Ross, Jadidi, Maani Ghaffari, Gan, Lu, Huang, Jiunn-Kai, Grizzle, Jessy W., and Eustice, Ryan M. 2018b. Hybrid Contact Preintegration for Visual-Inertial-Contact State Estimation Using Factor Graphs. Pages 3783–3790 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [440] Hartley, Ross, Mangelson, Josh, Gan, Lu, Ghaffari Jadidi, Maani, Walls, Jeffrey M., Eustice, Ryan M., and Grizzle, Jessy W. 2018c. Legged Robot State-Estimation Through Combined Forward Kinematic and Preintegrated Contact Factors. Pages 4422–4429 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [441] Hartley, Ross, Ghaffari, Maani, Eustice, Ryan M, and Grizzle, Jessy W. 2020. Contact-aided invariant extended Kalman filtering for robot state estimation. *Intl. J. of Robotics Research*, **39**(4), 402–430.
- [442] He, Dongjiao, Xu, Wei, Chen, Nan, Kong, Fanze, Yuan, Chongjian, and Zhang, Fu. 2023. Point-LIO: Robust High-Bandwidth Light Detection and Ranging Inertial Odometry. *Advanced Intelligent Systems*, **5**(7), 2200459.
- [443] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. 2016a. Deep residual learning for image recognition. Pages 770–778 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [444] He, Kaiming, Gkioxari, Georgia, Dollár, Piotr, and Girshick, Ross. 2017. Mask r-cnn. Pages 2961–2969 of: *Intl. Conf. on Computer Vision (ICCV)*.

- [445] He, Li, Wang, Xiaolong, and Zhang, Hong. 2016b. M2DP: A novel 3D point cloud descriptor and its application in loop closure detection. Pages 231–237 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [446] Heck, Martijn J.R. 2017. Highly integrated optical phased arrays: photonic integrated circuits for optical beam shaping and beam steering. *Nanophotonics*, **6**(1), 93–107.
- [447] Hendrycks, Dan, and Gimpel, Kevin. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *arXiv preprint*.
- [448] Henein, Mina, Zhang, Jun, Mahony, Robert, and Ila, Viorela. 2020. Dynamic SLAM: The need for speed. Pages 2123–2129 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [449] Henning, Dorian F, Laidlow, Tristan, and Leutenegger, Stefan. 2022. BodySLAM: joint camera localisation, mapping, and human motion tracking. Pages 656–673 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [450] Henning, Dorian F, Choi, Christopher, Schaefer, Simon, and Leutenegger, Stefan. 2023. BodySLAM++: Fast and tightly-coupled visual-inertial camera and human motion tracking. Pages 3781–3788 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [451] Herath, Sachini, Yan, Hang, and Furukawa, Yasutaka. 2020. Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods. Pages 3146–3152 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [452] Herath, Sachini, Caruso, David, Liu, Chen, Chen, Yufan, and Furukawa, Yasutaka. 2022. Neural inertial localization. Pages 6604–6613 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [453] Hermann, R., and Krener, A. 1977. Nonlinear controllability and observability. *IEEE Trans. on Automatic Control*, **22**(5), 728–740.
- [454] Herraez, Daniel Casado, Chang, Le, Zeller, Matthias, Wiesmann, Louis, Behley, Jens, Heidingsfeld, Michael, and Stachniss, Cyrill. 2024. SPR: Single-Scan Radar Place Recognition. *IEEE Robotics and Automation Letters*.
- [455] Hesch, J. A., Mirzaei, F. M., Mariottini, G. L., and Roumeliotis, S. I. 2010 (May). A Laser-Aided Inertial Navigation System (L-INS) for human localization in unknown indoor environments. Pages 5376–5382 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [456] Hesch, J.A., Kottas, D.G., Bowman, S.L., and Roumeliotis, S.I. 2013a. Consistency Analysis and Improvement of Vision-aided Inertial Navigation. *IEEE Trans. Robotics*, **30**(1), 158–176.
- [457] Hesch, JoelA., Kottas, DimitriosG., Bowman, SeanL., and Roumeliotis, StergiosI. 2013b. Towards Consistent Vision-Aided Inertial Navigation. Pages 559–574 of: Frazzoli, Emilio, Lozano-Perez, Tomas, Roy, Nicholas, and Rus, Daniela (eds), *Algorithmic Foundations of Robotics X*. Springer Tracts in Advanced Robotics, vol. 86. Springer Berlin Heidelberg.
- [458] Hess, Wolfgang, Kohler, Damon, Rapp, Holger, and Andor, Daniel. 2016. Real-time loop closure in 2D LIDAR SLAM. Pages 1271–1278 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [459] Hestenes, Magnus R, and Stiefel, Eduard. 1952. Methods of conjugate gradients for solving. *Journal of research of the National Bureau of Standards*, **49**(6), 409.

- [460] Hidalgo-Carrió, Javier, Gehrig, Daniel, and Scaramuzza, Davide. 2020 (Nov.). Learning Monocular Dense Depth from Events. Pages 534–542 of: *Intl. Conf. on 3D Vision (3DV)*.
- [461] Hidalgo-Carrió, Javier, Gallego, Guillermo, and Scaramuzza, Davide. 2022 (June). Event-aided Direct Sparse odometry. Pages 5781–5790 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [462] Himstedt, Marian, Frost, Jan, Hellbach, Sven, Böhme, Hans-Joachim, and Maehle, Erik. 2014. Large scale place recognition in 2D LIDAR scans using geometrical landmark relations. Pages 5030–5035 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [463] Ho, Jonathan, Jain, Ajay, and Abbeel, Pieter. 2020. Denoising Diffusion Probabilistic Models. Pages 6840–6851 of: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., and Lin, H. (eds), *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc.
- [464] Hoeller, David, Rudin, Nikita, Sako, Dhionis V., and Hutter, Marco. 2024. ANYmal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, **9**(88).
- [465] Holmes, Connor, and Barfoot, Timothy D. 2023. An efficient global optimality certificate for landmark-based SLAM. *IEEE Robotics and Automation Letters*, **8**(3), 1539–1546.
- [466] Holmes, Connor, Dümbgen, Frederike, and Barfoot, Timothy D. 2024. *On Semidefinite Relaxations for Matrix-Weighted State-Estimation Problems in Robotics*.
- [467] Holmstrom, Sven T. S., Baran, Utku, and Urey, Hakan. 2014. MEMS Laser Scanners: A Review. *Journal of Microelectromechanical Systems*, **23**(2), 259–275.
- [468] Honerkamp, Daniel, Büchner, Martin, Despinoy, Fabien, Welschehold, Tim, and Valada, Abhinav. 2024. Language-Grounded Dynamic Scene Graphs for Interactive Object Search with Mobile Manipulation. *IEEE Robotics and Automation Letters*.
- [469] Hong, Je Hyeong, Zach, Christopher, Fitzgibbon, Andrew, and Cipolla, Roberto. 2016. Projective bundle adjustment from arbitrary initialization using the variable projection method. Pages 477–493 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [470] Hong, Je Hyeong, Zach, Christopher, and Fitzgibbon, Andrew. 2017. Revisiting the variable projection method for separable nonlinear least squares problems. Pages 5939–5947 of: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- [471] Hong, Sheng, and et al. 2024. LIV-GaussMap: LiDAR-Inertial-Visual Fusion for Real-time 3D Radiance Field Map Rendering. *IEEE Robotics and Automation Letters*.
- [472] Hong, Yining, Lin, Chunru, Du, Yilun, Chen, Zhenfang, Tenenbaum, Joshua B, and Gan, Chuang. 2023a. 3D concept learning and reasoning from multi-view images. Pages 9202–9212 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [473] Hong, Ziyang, Petillot, Yvan, and Wang, Sen. 2020. RadarSLAM: Radar based Large-Scale SLAM in All Weathers. Pages 5164–5170 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

- [474] Hong, Ziyang, Petillot, Yvan, Wallace, Andrew, and Wang, Sen. 2022. RadarSLAM: A robust simultaneous localization and mapping system for all weather conditions. *Intl. J. of Robotics Research*, **41**(5), 519–542.
- [475] Hong, Ziyang, Petillot, Yvan, Zhang, Kaicheng, Xu, Shida, and Wang, Sen. 2023b. Large-Scale Radar Localization using Online Public Maps. Pages 3990–3996 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [476] Hoppe, Hugues, DeRose, Tony, Duchamp, Tom, McDonald, John, and Stuetzle, Werner. 1992. Surface reconstruction from unorganized points. Pages 71–78 of: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*. Association for Computing Machinery.
- [477] Horn, Berthold K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, **4**(4), 629–642.
- [478] Hornung, Armin, Wurm, Kai M., Bennewitz, Maren, Stachniss, Cyrill, and Burgard, Wolfram. 2013. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 189–206.
- [479] Hsiao, Ming, and Kaess, Michael. 2019. MH-iSAM2: Multi-hypothesis iSAM using Bayes Tree and Hypo-tree. Pages 1274–1280 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [480] Hu, Mu, Yin, Wei, Zhang, Chi, Cai, Zhipeng, Long, Xiaoxiao, Chen, Hao, Wang, Kaixuan, Yu, Gang, Shen, Chunhua, and Shen, Shaojie. 2024. Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. *IEEE Trans. Pattern Anal. Machine Intell.*
- [481] Hu, Yuhuang, Liu, Shih-Chii, and Delbruck, Tobi. 2021. v2e: From Video Frames to Realistic DVS Events. Pages 1312–1321 of: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops*.
- [482] Huai, Jianzhu, Wang, Binliang, Zhuang, Yuan, Chen, Yiwen, Li, Qipeng, Han, Yulong, and Toth, Charles. 2024. Snail-Radar: A large-scale diverse dataset for the evaluation of 4D-radar-based SLAM systems. *arXiv preprint*.
- [483] Huang, Binbin, Yu, Zehao, Chen, Anpei, Geiger, Andreas, and Gao, Shenghua. 2024a. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*. Association for Computing Machinery.
- [484] Huang, Chen, Mees, Oier, Zeng, Andy, and Burgard, Wolfram. 2022a. Visual Language Maps for Robot Navigation. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 10608–10615.
- [485] Huang, Guoquan. 2017. Towards Consistent Filtering for Discrete-Time Partially-Observable Nonlinear Systems. *Systems and Control Letters*, **106**(Aug.), 87–95.
- [486] Huang, Guoquan, Mourikis, Anastasios I., and Roumeliotis, Stergios I. 2010. Observability-based Rules for Designing Consistent EKF SLAM Estimators. *Intl. J. of Robotics Research*, **29**(5), 502–528.
- [487] Huang, Guoquan, Mourikis, Anastasios I., and Roumeliotis, Stergios I. 2011 (Sept.). An Observability Constrained Sliding Window Filter for SLAM. Pages 65–72 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [488] Huang, Huajian, Li, Longwei, Hui, Cheng, and Yeung, Sai-Kit. 2024b. PhotoSLAM: Real-time Simultaneous Localization and Photorealistic Mapping for

- Monocular, Stereo, and RGB-D Cameras. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [489] Huang, Jiahui, Zhou, Qunjie, Rabeti, Hesam, Korovko, Aleksandr, Ling, Huan, Ren, Xuanchi, Shen, Tianchang, Gao, Jun, Slepichev, Dmitry, Lin, Chen-Hsuan, et al. 2025. ViPE: Video Pose Engine for 3D Geometric Perception. *arXiv preprint arXiv:2508.10934*.
- [490] Huang, Jui-Te, Xu, Ruoyang, Hinduja, Akshay, and Kaess, Michael. 2024c (May). Multi-Radar Inertial Odometry for 3D State Estimation using mmWave Imaging Radar. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [491] Huang, Qiangqiang, Pu, Can, Khosoussi, Kasra, Rosen, David M, Fourie, Dehann, How, Jonathan P, and Leonard, John J. 2022b. Incremental non-Gaussian inference for SLAM using normalizing flows. *IEEE Trans. Robotics*, **39**(2), 1458–1475.
- [492] Huang, Shoudong, Chen, Yongbo, Zhao, Liang, Zhang, Yanhao, and Xu, Mengya. 2021. Some research questions for slam in deformable environments. Pages 7653–7660 of: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [493] Huang, Yewei, Shan, Tixiao, Chen, Fanfei, and Englot, Brendan. 2022c. DiSCo-SLAM: Distributed Scan Context-Enabled Multi-Robot LiDAR SLAM With Two-Stage Global-Local Graph Optimization. *IEEE Robotics and Automation Letters*, **7**(2), 1150–1157.
- [494] Huang, Zhaoyang, Shi, Xiaoyu, Zhang, Chao, Wang, Qiang, Cheung, Ka Chun, Qin, Hongwei, Dai, Jifeng, and Li, Hongsheng. 2022d. Flowformer: A transformer architecture for optical flow. Pages 668–685 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [495] Huber, P. 1981. *Robust Statistics*. John Wiley & Sons, New York, NY.
- [496] Hughes, Nathan, Chang, Yun, Hu, Siyi, Talak, Rajat, Abdulhai, Rumaia, Strader, Jared, and Carbone, Luca. 2024. Foundations of spatial perception for robotics: Hierarchical representations and real-time systems. *Intl. J. of Robotics Research*, **43**(10), 1457–1505.
- [497] Hwangbo, Jemin, Bellicoso, Carmine Dario, Fankhauser, Péter, and Hutter, Marco. 2016. Probabilistic foot contact estimation by fusing information from dynamics and differential/forward kinematics. Pages 3872–3878 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [498] Hwangbo, Jemin, Lee, Joonho, Dosovitskiy, Alexey, Bellicoso, Dario, Tsounis, Vassilios, Koltun, Vladlen, and Hutter, Marco. 2019. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, **4**(26).
- [499] Hyypä, J., Hyypä, H., Leckie, D., Gougeon, F., Yu, X., and Maltamo, M. 2008. Review of methods of small-footprint airborne laser scanning for extracting forest inventory data in boreal forests. *Intl. J. of Remote Sensing*, **29**(5), 1339–1366.
- [500] ICRA Quadruped Robot Challenge. 2024. *ICRA Quadruped Robot Challenge*. <https://quadruped-robot-challenges.notion.site/Quadruped-Robot-Challenges-bdc4af35638c4036817c3212e602b0e3>. [Online; accessed 10-Jun-2024].
- [501] Iglesias, José Pedro, Olsson, Carl, and Kahl, Fredrik. 2020. Global Optimality for Point Set Registration Using Semidefinite Programming. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [502] Ilg, Eddy, Mayer, Nikolaus, Saikia, Tonmoy, Keuper, Margret, Dosovitskiy, Alexey, and Brox, Thomas. 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. Pages 2462–2470 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [503] Indelman, V., Wiliams, S., Kaess, M., and Dellaert, F. 2012. Factor Graph Based Incremental Smoothing in Inertial Navigation Systems. In: *Intl. Conf. on Information Fusion (FUSION)*.
- [504] Innmann, Matthias, Zollhöfer, Michael, Nießner, Matthias, Theobalt, Christian, and Stamminger, Marc. 2016. VolumeDeform: Real-time volumetric non-rigid reconstruction. Pages 362–379 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [505] Ivan, Jean-Paul A, Stoyanov, Todor, and Stork, Johannes A. 2022. Online Distance Field Priors for Gaussian Process Implicit Surfaces. *IEEE Robotics and Automation Letters*, **7**(4), 8996–9003.
- [506] Izadi, Shahram, Kim, David, Hilliges, Otmar, Molyneaux, David, Newcombe, Richard, Kohli, Pushmeet, Shotton, Jamie, Hodges, Steve, Freeman, Dustin, Davison, Andrew, et al. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. Pages 559–568 of: *ACM Symp. on User interface software and technology*.
- [507] Izatt, G., Dai, H., and Tedrake, R. 2017. Globally Optimal Object Pose Estimation in Point Clouds with Mixed-Integer Programming. In: *Intl. Symp. of Robotics Research (ISRR)*.
- [508] Izquierdo, Sergio, and Civera, Javier. 2024. Optimal transport aggregation for visual place recognition. Pages 17658–17668 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [509] Jaimez, M., Kerl, C., Gonzalez-Jimenez, J., and Cremers, D. 2017. Fast Odometry and Scene Flow from RGB-D Cameras based on Geometric Clustering. In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.
- [510] Jang, Hyesu, Jung, Minwoo, and Kim, Ayoung. 2023. RaPlace: Place Recognition for Imaging Radar using Radon Transform and Mutable Threshold. Pages 11194–11201 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [511] Jatavallabhula, Krishna Murthy, Iyer, Ganesh, and Paull, Liam. 2020.  $\nabla$  slam: Dense slam meets automatic differentiation. Pages 2130–2137 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [512] Jatavallabhula, Krishna Murthy, Kuwajerwala, Alihusein, Gu, Qiao, Omama, Mohd, Chen, Tao, Maalouf, Alaa, Li, Shuang, Iyer, Ganesh, Saryazdi, Soroush, Keetha, Nikhil, et al. 2023. Conceptfusion: Open-set multimodal 3d mapping. *arXiv preprint*.
- [513] Jenelten, Fabian, Hwangbo, Jemin, Tresoldi, Fabian, Bellicoso, C. Dario, and Hutter, Marco. 2019. Dynamic Locomotion on Slippery Ground. *IEEE Robotics and Automation Letters*, **4**(4), 4170–4176.
- [514] Jeong, Jinyong, Cho, Younggun, Shin, Young-Sik, Roh, Hyunchul, and Kim, Ayoung. 2019. Complex urban dataset with multi-level sensors from highly diverse urban environments. *Intl. J. of Robotics Research*, **38**(6), 642–657.
- [515] Ji, Gwanghyeon, Mun, Juhyeok, Kim, Hyeongjun, and Hwangbo, Jemin. 2022. Concurrent Training of a Control Policy and a State Estimator for

- Dynamic and Robust Legged Locomotion. *IEEE Robotics and Automation Letters*, **7**(2), 4630–4637.
- [516] Ji, Kaiyi, Yang, Junjie, and Liang, Yingbin. 2021. Bilevel optimization: Convergence analysis and enhanced design. Pages 4882–4892 of: *Intl. Conf. on Machine Learning (ICML)*. PMLR.
- [517] Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. 2014. Caffe: Convolutional architecture for fast feature embedding. Pages 675–678 of: *22nd ACM international conference on Multimedia*.
- [518] Jiang, Fan, Agrawal, Varun, Buchanan, Russell, Fallon, Maurice, and Dellaert, Frank. 2021. iMHS: An incremental multi-hypothesis smoother. *arXiv preprint*.
- [519] Jiang, H., Huang, B., Wu, R., Li, Z., Garg, S., Nayyeri, H., Wang, S., and Li, Y. 2024. RoboEXP: Action-Conditioned Scene Graph via Interactive Exploration for Robotic Manipulation. In: *arXiv preprint*.
- [520] Jiang, Huaizu, Sun, Deqing, Jampani, Varun, Yang, Ming-Hsuan, Learned-Miller, Erik, and Kautz, Jan. 2018. Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation. Pages 9000–9008 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [521] Jiang, Kun, Gao, Shuang, Zhang, Xudong, Li, Jijunnan, Guo, Yandong, Liu, Shijie, Li, Chunlai, and Wang, Jianyu. 2023. SELVO: A Semantic-Enhanced Lidar-Visual Odometry. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [522] Jiao, Jianhao, Wei, Hexiang, Hu, Tianshuai, Hu, Xiangcheng, Zhu, Yilong, He, Zhijian, Wu, Jin, Yu, Jingwen, Xie, Xupeng, Huang, Huaiyang, Geng, Ruoyu, Wang, Lujia, and Liu, Ming. 2022. FusionPortable: A Multi-Sensor Campus-Scene Dataset for Evaluation of Localization and Mapping Accuracy on Diverse Platforms. Pages 3851–3856 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [523] Jin, Hailin, Favaro, Paolo, and Soatto, Stefano. 2000. Real-time 3D motion and structure of point features: a front-end system for vision-based control and interaction. Pages 778–779 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol. 2.
- [524] Johari, Mohammad Mahdi, Carta, Camilla, and Fleuret, François. 2023. Eslam: Efficient dense slam system based on hybrid representation of signed distance fields. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [525] Johnson, J, Mangelson, J, Barfoot, T D, and Beard, R. 2024. *Continuous-time Trajectory Estimation: A Comparative Study Between Gaussian Process and Spline-based Approaches*.
- [526] Judd, Kevin M, Gammell, Jonathan D, and Newman, Paul. 2018. Multimotion visual odometry (MVO): Simultaneous estimation of camera and third-party motions. Pages 3949–3956 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [527] Jumper, John, Evans, Richard, Pritzel, Alexander, Green, Tim, Figurnov, Michael, Ronneberger, Olaf, Tunyasuvunakool, Kathryn, Bates, Russ, Žídek, Augustin, Potapenko, Anna, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature*, **596**(7873), 583–589.

- [528] Jung, Minwoo, Yang, Wooseong, Lee, Dongjae, Gil, Hyeonjae, Kim, Giseop, and Kim, Ayoung. 0. HeLiPR: Heterogeneous LiDAR dataset for inter-LiDAR place recognition under spatiotemporal variations. *Intl. J. of Robotics Research*, **0**(0), 02783649241242136.
- [529] Kabalar, Julia, Wu, Shun-Cheng, Wald, Johanna, Tateno, Keisuke, Navab, Nassir, and Tombari, Federico. 2023. Towards Long-Term Retrieval-Based Visual Localization in Indoor Environments With Changes. *IEEE Robotics and Automation Letters*, **8**(4), 1975–1982.
- [530] Kaess, M., Ranganathan, A., and Dellaert, F. 2008. iSAM: Incremental Smoothing and Mapping. *IEEE Trans. Robotics*, **24**(6), 1365–1378.
- [531] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. 2011 (May). iSAM2: Incremental Smoothing and Mapping with Fluid Relinearization and Incremental Variable Reordering. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [532] Kaess, Michael. 2015. Simultaneous localization and mapping with infinite planes. Pages 4605–4611 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [533] Kaess, Michael, Johannsson, Hordur, Roberts, Richard, Ila, Viorela, Leonard, John J, and Dellaert, Frank. 2012. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, **31**(2), 216–235.
- [534] Kaiser, Jacques, Tieck, J. Camillo Vasquez, Hubschneider, Christian, Wolf, Peter, Weber, Michael, Hoff, Michael, Friedrich, Alexander, Wojtasik, Konrad, Roennau, Arne, Kohlhaas, Ralf, Dillmann, Rüdiger, and Zöllner, J. Marius. 2016. Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. Pages 127–134 of: *IEEE Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*.
- [535] Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schröder, J., Thuy, M., Goebel, M., v. Hundelshausen, F., Pink, O., Frese, C., and Stiller, C. 2008. Team AnniWay’s Autonomous System for the 2007 DARPA Urban Challenge. *J. of Field Robotics*, 615–639.
- [536] Kannala, Juho, and Brandt, Sami S. 2006. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Trans. Pattern Anal. Machine Intell.*, **28**(8), 1335–1340.
- [537] Karimian, A., and Tron, R. 2023. Essential Matrix Estimation using Convex Relaxations in Orthogonal Space. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [538] Kassab, Christina, Mattamala, Matías, Morin, Sacha, Büchner, Martin, Valada, Abhinav, Paull, Liam, and Fallon, Maurice. 2024a. The Bare Necessities: Designing Simple, Effective Open-Vocabulary Scene Graphs. *arXiv preprint*.
- [539] Kassab, Christina, Mattamala, Matias, Zhang, Lintong, and Fallon, Maurice. 2024b. Language-extended indoor slam (lexis): A versatile system for real-time visual scene understanding. Pages 15988–15994 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [540] Kato, Hiroharu, Ushiku, Yoshitaka, and Harada, Tatsuya. 2018. Neural 3d mesh renderer. Pages 3907–3916 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [541] Kato, Hiroharu, Beker, Deniz, Morariu, Mihai, Ando, Takahiro, Matsuoka, Toru, Kehl, Wadim, and Gaidon, Adrien. 2020. Differentiable rendering: A survey. *arXiv preprint*.
- [542] Katragadda, Saimouli, Wu, Cho-Ying, Guo, Yuliang, Xinyu Huang, Guoquan Huang, and Ren, Liu. 2025. Online Language Splatting. In: *arXiv preprint*.
- [543] Katz, Benjamin, Carlo, Jared Di, and Kim, Sangbae. 2019. Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control. Pages 6295–6301 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [544] Kazhdan, Michael, Bolitho, Matthew, and Hoppe, Hugues. 2006. Poisson surface reconstruction. Pages 61–70 of: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Eurographics Association.
- [545] Keenan Burnett, Angela P. Schoellig, Timothy D. Barfoot. 2021. Do We Need to Compensate for Motion Distortion and Doppler Effects in Spinning Radar Navigation? *IEEE Robotics and Automation Letters*, **6**(2), 771–778.
- [546] Keetha, Nikhil, Mishra, Avneesh, Karhade, Jay, Jatavallabhula, Krishna Murthy, Scherer, Sebastian, Krishna, Madhava, and Garg, Sourav. 2023. Anyloc: Towards universal visual place recognition. *IEEE Robotics and Automation Letters*.
- [547] Keetha, Nikhil, Karhade, Jay, Jatavallabhula, Krishna Murthy, Yang, Gengshan, Scherer, Sebastian, Ramanan, Deva, and Luiten, Jonathon. 2024. Splatam: Splat track & map 3d gaussians for dense rgb-d slam. Pages 21357–21366 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [548] Keller, Maik, Lefloch, Damien, Lambers, Martin, Izadi, Shahram, Weyrich, Tim, and Kolb, Andreas. 2013. Real-time 3d reconstruction in dynamic scenes using point-based fusion. Pages 1–8 of: *Intl. Conf. on 3D Vision (3DV)*. IEEE.
- [549] Kellner, Dominik, Klappstein, Jens, and Dietmayer, Klaus. 2012. Grid-based DBSCAN for clustering extended objects in radar data. Pages 365–370 of: *IEEE Intelligent Vehicles Symposium (IV)*.
- [550] Kellner, Dominik, Barjenbruch, Michael, Klappstein, Jens, Dickmann, Jürgen, and Dietmayer, Klaus. 2013. Instantaneous ego-motion estimation using Doppler radar. Pages 869–874 of: *IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*.
- [551] Kerbl, Bernhard, Kopanas, Georgios, Leimkühler, Thomas, and Drettakis, George. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, **42**(4), 139–1.
- [552] Kerl, Christian, Sturm, Jürgen, and Cremers, Daniel. 2013a. Dense visual SLAM for RGB-D cameras. Pages 2100–2106 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [553] Kerl, Christian, Sturm, Jürgen, and Cremers, Daniel. 2013b. Robust odometry estimation for RGB-D cameras. Pages 3748–3754 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [554] Kerr, Justin, Kim, Chung Min, Goldberg, Ken, Kanazawa, Angjoo, and Tancik, Matthew. 2023. LERF: Language Embedded Radiance Fields. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [555] Keselman, Leonid, and Hebert, Martial. 2022. Approximate differentiable

- rendering with algebraic surfaces. Pages 596–614 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [556] Khader, Motaz, and Cherian, Samir. 2020. An introduction to automotive lidar. *Texas Instruments*.
- [557] Khattak, Shehryar, Nguyen, Huan, Mascarich, Frank, Dang, Tung, and Alexis, Kostas. 2020. Complementary multi-modal sensor fusion for resilient robot pose estimation in subterranean environments. Pages 1024–1029 of: *Intl. Conf. on Unmanned Aircraft Systems (ICUAS)*.
- [558] Khazatsky, Alexander, Pertsch, Karl, Nair, Suraj, Balakrishna, Ashwin, Dasari, Sudeep, Karamcheti, Siddharth, Nasiriany, Soroush, Srirama, Mohan Kumar, Chen, Lawrence Yunliang, Ellis, Kirsty, Fagan, Peter David, Hejna, Joey, Itkina, Masha, Lepert, Marion, Ma, Yecheng Jason, Miller, Patrick Tree, Wu, Jimmy, Belkhale, Suneel, Dass, Shivin, Ha, Huy, Jain, Arhan, Lee, Abraham, Lee, Youngwoon, Memmel, Marius, Park, Sungjae, Radosavovic, Ilija, Wang, Kaiyuan, Zhan, Albert, Black, Kevin, Chi, Cheng, Hatch, Kyle Beltran, Lin, Shan, Lu, Jingpei, Mercat, Jean, Rehman, Abdul, Sanketi, Pannag R, Sharma, Archit, Simpson, Cody, Vuong, Quan, Walke, Homer Rich, Wulfe, Blake, Xiao, Ted, Yang, Jonathan Heewon, Yavary, Arefeh, Zhao, Tony Z., Agia, Christopher, Baijal, Rohan, Castro, Mateo Guaman, Chen, Daphne, Chen, Qiuyu, Chung, Trinity, Drake, Jaimyn, Foster, Ethan Paul, Gao, Jensen, Herrera, David Antonio, Heo, Minho, Hsu, Kyle, Hu, Jiaheng, Jackson, Donovon, Le, Charlotte, Li, Yunshuang, Lin, Kevin, Lin, Roy, Ma, Zehan, Maddukuri, Abhiram, Mirchandani, Suvir, Morton, Daniel, Nguyen, Tony, O'Neill, Abigail, Scalise, Rosario, Seale, Derick, Son, Victor, Tian, Stephen, Tran, Emi, Wang, Andrew E., Wu, Yilin, Xie, Annie, Yang, Jingyun, Yin, Patrick, Zhang, Yunchu, Bastani, Osbert, Berseth, Glen, Bohg, Jeannette, Goldberg, Ken, Gupta, Abhinav, Gupta, Abhishek, Jayaraman, Dinesh, Lim, Joseph J, Malik, Jitendra, Martín-Martín, Roberto, Ramamoorthy, Subramanian, Sadigh, Dorsa, Song, Shuran, Wu, Jiajun, Yip, Michael C., Zhu, Yu, Kollar, Thomas, Levine, Sergey, and Finn, Chelsea. 2024. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. *arXiv preprint*.
- [559] Khosoussi, Kasra, Huang, Shoudong, and Dissanayake, Gamini. 2016. Tree-connectivity: Evaluating the graphical structure of SLAM. Pages 1316–1322 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [560] Khosoussi, Kasra, Giamou, Matthew, Sukhatme, Gaurav S, Huang, Shoudong, Dissanayake, Gamini, and How, Jonathan P. 2019. Reliable graphs for SLAM. *Intl. J. of Robotics Research*, **38**(2-3), 260–298.
- [561] Kim, Been, Kaess, Michael, Fletcher, Luke, Leonard, John, Bachrach, Abraham, Roy, Nicholas, and Teller, Seth. 2010. Multiple relative pose graphs for robust cooperative mapping. Pages 3185–3192 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [562] Kim, Giseop, and Kim, Ayoung. 2018. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. Pages 4802–4809 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [563] Kim, Giseop, and Kim, Ayoung. 2020. Remove, then revert: Static point cloud map construction using multiresolution range images. Pages 10758–

- 10765 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [564] Kim, Giseop, and Kim, Ayoung. 2022. Lt-mapper: A modular framework for lidar-based lifelong mapping. Pages 7995–8002 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [565] Kim, Giseop, Park, Byungjae, and Kim, Ayoung. 2019a. 1-day learning, 1-year localization: Long-term lidar localization using scan context image. *IEEE Robotics and Automation Letters*, **4**(2), 1948–1955.
- [566] Kim, Giseop, Park, Yeong Sang, Cho, Younghun, Jeong, Jinyong, and Kim, Ayoung. 2020. Mulran: Multimodal range dataset for urban place recognition. Pages 6246–6253 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [567] Kim, Giseop, Choi, Sunwook, and Kim, Ayoung. 2021. Scan context++: Structural place recognition robust to rotation and lateral variations in urban environments. *IEEE Trans. Robotics*, **38**(3), 1856–1874.
- [568] Kim, Hanjun, Jung, Minwoo, Noh, Chiyun, Jung, Sangwoo, Song, Hyunho, Yang, Wooseong, Jang, Hyesu, and Kim, Ayoung. 2025. HeRCULES: Heterogeneous Radar Dataset in Complex Urban Environment for Multi-session Radar SLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [569] Kim, Hanme, Handa, Ankur, Benosman, Ryad, Ieng, Sio-Hoi, and Davison, Andrew J. 2014. Simultaneous Mosaicing and Tracking with an Event Camera. In: *British Machine Vision Conf. (BMVC)*.
- [570] Kim, Hanme, Leutenegger, Stefan, and Davison, Andrew J. 2016. Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera. Pages 349–364 of: *European Conf. on Computer Vision (ECCV)*.
- [571] Kim, Haram, and Kim, H. Jin. 2021. Real-Time Rotational Motion Estimation With Contrast Maximization Over Globally Aligned Events. *IEEE Robotics and Automation Letters*, **6**(3), 6016–6023.
- [572] Kim, Moo Jin, Pertsch, Karl, Karamcheti, Siddharth, Xiao, Ted, Balakrishna, Ashwin, Nair, Suraj, Rafailov, Rafael, Foster, Ethan, Lam, Grace, Sanketi, Pannag, Vuong, Quan, Kollar, Thomas, Burchfiel, Benjamin, Tedrake, Russ, Sadigh, Dorsa, Levine, Sergey, Liang, Percy, and Finn, Chelsea. 2024. OpenVLA: An Open-Source Vision-Language-Action Model. *arXiv preprint*.
- [573] Kim, S., and Kim, J. 2012. Building occupancy maps with a mixture of Gaussian processes. Pages 4756–4761 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [574] Kim, S., and Kim, J. 2013. Occupancy Mapping and Surface Reconstruction Using Local Gaussian Processes With Kinect Sensors. Pages 1335–1346 of: *IEEE Trans. on Cybernetics*.
- [575] Kim, Soohwan, and Kim, Jonghyuk. 2015. *GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes*. Springer International Publishing. Pages 319–332.
- [576] Kim, Ue-Hwan, Park, Jin-Man, Song, Taek-Jin, and Kim, Jong-Hwan. 2019b. 3-D Scene Graph: A Sparse and Semantic Representation of Physical Environments for Intelligent Agents. *IEEE Trans. on Cybernetics*, **PP**(Aug.), 1–13.
- [577] Kim, Yeeun, Yu, Byeongho, Lee, Eungchang Mason, Kim, Joon-ha, Park, Hae-won, and Myung, Hyun. 2022. STEP: State Estimator for Legged

- Robots Using a Preintegrated Foot Velocity Factor. *IEEE Robotics and Automation Letters*, **7**(2), 4456–4463.
- [578] Kingma, Diederik P., and Ba, Jimmy. 2014. Adam: A method for stochastic optimization. *arXiv preprint*.
- [579] Kirillov, Alexander, He, Kaiming, Girshick, Ross, Rother, Carsten, and Dollár, Piotr. 2019 (June). Panoptic Segmentation. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [580] Kirillov, Alexander, Mintun, Eric, Ravi, Nikhila, Mao, Hanzi, Rolland, Chloe, Gustafson, Laura, Xiao, Tete, Whitehead, Spencer, Berg, Alexander C., Lo, Wan-Yen, et al. 2023. Segment anything. Pages 4015–4026 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [581] Kitanov, Andrej, and Indelman, Vadim. 2024. Topological belief space planning for active SLAM with pairwise Gaussian potentials and performance guarantees. *Intl. J. of Robotics Research*, **43**(1), 69–97.
- [582] Klein, Georg, and Murray, David. 2007. Parallel tracking and mapping for small AR workspaces. Pages 225–234 of: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. IEEE.
- [583] Klenk, Simon, Chui, Jason, Demmel, Nikolaus, and Cremers, Daniel. 2021. TUM-VIE: The TUM Stereo Visual-Inertial Event Dataset. Pages 8601–8608 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [584] Klenk, Simon, Motzett, Marvin, Koestler, Lukas, and Cremers, Daniel. 2024. Deep Event Visual Odometry. Pages 739–749 of: *Intl. Conf. on 3D Vision (3DV)*.
- [585] Kobayashi, Sosuke, Matsumoto, Eiichi, and Sitzmann, Vincent. 2022. De-composing nerf for editing via feature field distillation. *Advances in Neural Information Processing Systems (NIPS)*.
- [586] Koch, Sebastian, Hermosilla, Pedro, Vaskevicius, Narunas, Colosi, Mirco, and Ropinski, Timo. 2024a. Lang3DSG: Language-based contrastive pre-training for 3D Scene Graph prediction. In: *Intl. Conf. on 3D Vision (3DV)*.
- [587] Koch, Sebastian, Vaskevicius, Narunas, Colosi, Mirco, Hermosilla, Pedro, and Ropinski, Timo. 2024b (June). Open3DSG: Open-Vocabulary 3D Scene Graphs from Point Clouds with Queryable Objects and Open-Set Relationships. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [588] Koch, Sebastian, Hermosilla, Pedro, Vaskevicius, Narunas, Colosi, Mirco, and Ropinski, Timo. 2024c (January). SGRec3D: Self-Supervised 3D Scene Graph Learning via Object-Level Scene Reconstruction. Pages 3404–3414 of: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.
- [589] Koenemann, Jonas, Licitra, Giovanni, Alp, Mustafa, and Diehl, Moritz. 2017. *Openocl—open optimal control library*.
- [590] Koide, Kenji, Yokozuka, Masashi, Oishi, Shuji, and Banno, Atsuhiko. 2021. Voxelized gicp for fast and accurate 3d point cloud registration. Pages 11054–11059 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [591] Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.
- [592] Kong, Xin, Yang, Xuemeng, Zhai, Guangyao, Zhao, Xiangrui, Zeng, Xianfang, Wang, Mengmeng, Liu, Yong, Li, Wanlong, and Wen, Feng. 2020. Se-

- mantic Graph Based Place Recognition for 3D Point Clouds. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [593] Konolige, K. 2004. Large-scale map-making. In: *Proc. 21<sup>th</sup> AAAI National Conference on AI*.
- [594] Konolige, K., and Agrawal, M. 2008. FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping. *IEEE Trans. Robotics*, **24**, 1066–1077.
- [595] Konolige, Kurt, Grisetti, Giorgio, Kümmerle, Rainer, Burgard, Wolfram, Limketkai, Benson, and Vincent, Regis. 2010. Efficient sparse pose adjustment for 2D mapping. Pages 22–29 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [596] Kopanas, Georgios, Philip, Julien, Leimkühler, Thomas, and Drettakis, George. 2021. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, **40**(4).
- [597] Koskinen, Markku, Kostamovaara, Juha Tapiro, and Myllylae, Risto A. 1992. Comparison of continuous-wave and pulsed time-of-flight laser range-finding techniques. Pages 296–305 of: *Optics, Illumination, and Image Sensing for Machine Vision VI*, vol. 1614.
- [598] Kottas, Dimitrios, and Roumeliotis, Stergios. 2013 (June 24–28,). Exploiting Urban Scenes for Vision-aided Inertial Navigation. In: *Robotics: Science and Systems (RSS)*.
- [599] Kottege, Navinda, Scherer, Sebastian, Faigl, J., and Agha, Ali. 2022. Special Issue on Advancements and Lessons Learned during Phases I and II of the DARPA Subterranean Challenge. *Field Robotics*, 1947–1950.
- [600] Kottege, Navinda, Williams, Jason, Tidd, Brendan, Talbot, Fletcher, Steindl, Ryan, Cox, Mark, Frousheger, Dennis, Hines, Thomas, Pitt, Alex, Tam, Benjamin, Wood, Brett, Hanson, Lauren, Surdo, Katrina Lo, Molnar, Thomas, Wildie, Matt, Stepanas, Kazys, Catt, Gavin, Tychsen-Smith, Lachlan, Penfold, Dean, Overs, Leslie, Ramezani, Milad, Khosoussi, Kasra, Kendoul, Farid, Wagner, Glenn, Palmer, Duncan, Manderson, Jack, Medek, Corey, O'Brien, Matthew, Chen, Shengkang, and Arkin, Ronald C. 2024. Heterogeneous Robot Teams with Unified Perception and Autonomy: How Team CSIRO Data61 Tied for the Top Score at the DARPA Subterranean Challenge. *Field Robotics*, 313–359.
- [601] Krajník, Tomáš, Fentanes, Jaime Pulido, Hanheide, Marc, and Duckett, Tom. 2016. Persistent localization and life-long mapping in changing environments using the frequency map enhancement. Pages 4558–4563 of: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [602] Krajník, Tomáš, Fentanes, Jaime P, Santos, Joao M, and Duckett, Tom. 2017. Fremen: Frequency map enhancement for long-term mobile robot autonomy in changing environments. *IEEE Trans. Robotics*, **33**(4), 964–977.
- [603] Kramer, Andrew, and Heckman, Christoffer. 2020. Radar-Inertial State Estimation and Obstacle Detection for Micro-Aerial Vehicles in Dense Fog. Pages 3–16 of: *Intl. Sym. on Experimental Robotics (ISER)*. Springer.
- [604] Kramer, Andrew, Stahoviak, Carl, Santamaria-Navarro, Angel, Agha-Mohammadi, Ali-Akbar, and Heckman, Christoffer. 2020. Radar-inertial

- ego-velocity estimation for visually degraded environments. Pages 5739–5746 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [605] Kramer, Andrew, Harlow, Kyle, Williams, Christopher, and Heckman, Christoffer. 2022. ColoRadar: The Direct 3D Millimeter Wave Radar Dataset. *Intl. J. of Robotics Research*, **41**(4), 351–360.
- [606] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. 2012. Imagenet classification with deep convolutional neural networks. *Conf. Neural Information Processing Systems (NIPS)*, **25**.
- [607] Kubelka, Vladimír, Fritz, Emil, and Magnusson, Martin. 2024. Do we need scan-matching in radar odometry? In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [608] Kucner, Tomasz Piotr, Magnusson, Martin, Mghames, Sariah, Palmieri, Luigi, Verdoja, Francesco, Swaminathan, Chittaranjan Srinivas, Krajiník, Tomáš, Schaffernicht, Erik, Bellotto, Nicola, Hanheide, Marc, et al. 2023. Survey of maps of dynamics for mobile robots. *Intl. J. of Robotics Research*, **42**(11), 977–1006.
- [609] Kueng, Beat, Mueggler, Elias, Gallego, Guillermo, and Scaramuzza, Davide. 2016. Low-latency Visual Odometry using Event-based Feature Tracks. Pages 16–23 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [610] Kuipers, Benjamin. 2000. The Spatial Semantic Hierarchy. *Artificial Intelligence*, **119**, 191–233.
- [611] Kukko, Antero, Kaijaluoto, Risto, Kaartinen, Harri, Lehtola, Ville V, Jaakkola, Anttoni, and Hyppä, Juha. 2017. Graph SLAM correction for single scanner MLS forest data under boreal forest canopy. *ISPRS J. of Photogrammetry and Remote Sensing (JPRS)*, **132**, 199–209.
- [612] Kümmeler, Rainer, Grisetti, Giorgio, Strasdat, Hauke, Konolige, Kurt, and Burgard, Wolfram. 2011. G<sup>2</sup>o: A general framework for graph optimization. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [613] Kung, Pou-Chun, Wang, Chieh-Chih, and Lin, Wen-Chieh. 2021. A Normal Distribution Transform-Based Radar Odometry Designed For Scanning and Automotive Radars. Pages 14417–14423 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [614] Labb  , Mathieu, and Michaud, Fran  ois. 2019. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *J. of Field Robotics*, **36**(2), 416–446.
- [615] Laina, Sebastian Barbas, Boche, Simon, Papathodorou, Sotiris, Schaefer, Simon, Jung, Jaehyung, and Leutenegger, Stefan. 2025. FindAnything: Open-Vocabulary and Object-Centric Mapping for Robot Exploration in Any Environment. *arXiv preprint*.
- [616] Lajoie, P., Hu, S., Beltrame, G., and Carbone, L. 2019. Modeling Perceptual Aliasing in SLAM via Discrete-Continuous Graphical Models. *IEEE Robotics and Automation Letters*.
- [617] Lamarca, Jose, and Montiel, J.M.M. 2018. Camera Tracking for SLAM in Deformable Maps. In: *European Conf. on Computer Vision Workshops*.
- [618] Lamarca, Jose, Parashar, Shaifali, Bartoli, Adrien, and Montiel, J.M.M. 2021. DefSLAM: Tracking and Mapping of Deforming Scenes From Monocular Sequences. *IEEE Trans. Robotics*, **37**(1), 291–303.

- [619] Lamarca, José, Gómez Rodríguez, Juan J., Tardós, Juan D., and Montiel, J.M.M. 2022. Direct and Sparse Deformable Tracking. *IEEE Robotics and Automation Letters*, **7**(4), 11450–11457.
- [620] Landry, David, Pomerleau, Francois, and Giguere, Philippe. 2019. CELLO-3D: Estimating the Covariance of ICP in the Real World. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [621] Lang, Ben. 2022 (May). *Reality Labs Chief Scientist Outlines a New Compute Architecture for True AR Glasses*. <https://www.roadtovr.com/michael-abrash-iedm-2021-compute-architecture-for-ar-glasses/>.
- [622] Lang, Xiaolei, Li, Laijian, Wu, Chenming, Zhao, Chen, Liu, Lina, Liu, Yong, Lv, Jiajun, and Zuo, Xingxing. 2025. Gaussian-LIC: Real-Time Photo-Realistic SLAM with Gaussian Splatting and LiDAR-Inertial-Camera Fusion. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [623] Langer, Edith, Patten, Timothy, and Vincze, Markus. 2020. Robust and efficient object change detection by combining global semantic information and local geometric verification. Pages 8453–8460 of: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [624] Lasserre, J. 2010. *Moments, positive polynomials and their applications*. Vol. 1. World Scientific.
- [625] Lasserre, Jean B. 2001. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, **11**(3), 796–817.
- [626] Latif, Y., Lerma, C. D. C., and Neira, J. 2012. Robust Loop Closing Over Time. In: *Robotics: Science and Systems (RSS)*.
- [627] Lau, Boris, Sprunk, Christoph, and Burgard, Wolfram. 2013. Efficient grid-based spatial representations for robot navigation in dynamic environments. *J. on Robotics and Autonomous Systems (RAS)*, **61**(10), 1116–1130.
- [628] Le Gentil, C., and Vidal-Calleja, T. 2021. Continuous Integration over SO(3) for IMU Preintegration. In: *Robotics: Science and Systems (RSS)*.
- [629] Le Gentil, Cedric, Vidal-Calleja, Teresa, and Huang, Shoudong. 2018. 3D Lidar-IMU Calibration based on Upsampled Preintegrated Measurements for Motion Distortion Correction. Pages 2149–2155 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [630] Le Gentil, Cedric, Vidal-Calleja, Teresa, and Huang, Shoudong. 2020. Gaussian Process Preintegration for Inertial-Aided State Estimation. *IEEE Robotics and Automation Letters*, **5**(2), 2108–2114.
- [631] Le Gentil, Cedric, Vidal-Calleja, Teresa, and Huang, Shoudong. 2021. IN2LAAMA: Inertial Lidar Localization Autocalibration and Mapping. *IEEE Trans. Robotics*, **37**(1), 275–290.
- [632] Le Gentil, Cedric, Ouabi, Othmane-Latif, Wu, Lan, Pradalier, Cedric, and Vidal-Calleja, Teresa. 2023. Accurate Gaussian-Process-based Distance Fields with Applications to Echolocation and Mapping. *IEEE Robotics and Automation Letters*, 1–8.
- [633] Lee, Alex Junho, Cho, Younggun, Shin, Young-sik, Kim, Ayoung, and Myung, Hyun. 2022. ViViD++: Vision for Visibility Dataset. *IEEE Robotics and Automation Letters*, **7**(3), 6282–6289.
- [634] Lee, Bhoram, Zhang, Clark, Huang, Zonghao, and Lee, Daniel D. 2019. Online continuous mapping using gaussian process implicit surfaces. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [635] Lee, G. H., Fraundorfer, F., and Pollefeys, M. 2013. Robust pose-graph loop-closures with expectation-maximization. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [636] Lee, Joonho, Hwangbo, Jemin, Wellhausen, Lorenz, Koltun, Vladlen, and Hutter, Marco. 2020. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, **5**(47), 5986.
- [637] Leordeanu, Marius, and Hebert, Martial. 2005. A spectral technique for correspondence problems using pairwise constraints. Pages 1482–1489 of: *Intl. Conf. on Computer Vision (ICCV)*, vol. 2. IEEE.
- [638] Lepetit, Vincent, Moreno-Noguer, Francesc, and Fua, Pascal. 2009. Epnp: An accurate o (n) solution to the pnp problem. *Intl. J. of Computer Vision*, **81**(2), 155.
- [639] Lepora, Nathan F., and Lloyd, John. 2020. Optimal Deep Learning for Robot Touch: Training Accurate Pose Models of 3D Surfaces and Edges. *IEEE Robotics and Automation Magazine*, **27**(2), 66–77.
- [640] Leroy, Vincent, Cabon, Yohann, and Revaud, Jérôme. 2024. Grounding image matching in 3d with mast3r. Pages 71–91 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [641] Leutenegger, S., Chli, M., and Siegwart, R. 2011. BRISK: Binary Robust Invariance Scalable Keypoints. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [642] Leutenegger, Stefan. 2022. Okvis2: Realtime scalable visual-inertial slam with loop closure. *arXiv preprint*.
- [643] Leutenegger, Stefan, Lynen, Simon, Bosse, Michael, Siegwart, Roland, and Furgale, Paul. 2015. Keyframe-based visual–inertial odometry using nonlinear optimization. *Intl. J. of Robotics Research*, **34**(3), 314–334.
- [644] Levenberg, K. 1944. A Method for the Solution of Certain Nonlinear Problems in Least Squares. *Quart. Appl. Math.*, **2**(2), 164–168.
- [645] Li, Boyi, Weinberger, Kilian Q, Belongie, Serge, Koltun, Vladlen, and Ranftl, Rene. 2022. Language-driven Semantic Segmentation. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [646] Li, Chunyuan, Wong, Cliff, Zhang, Sheng, Usuyama, Naoto, Liu, Haotian, Yang, Jianwei, Naumann, Tristan, Poon, Hoifung, and Gao, Jianfeng. 2024a. Llava-med: Training a large language-and-vision assistant for biomedicine in one day. *Advances in Neural Information Processing Systems (NIPS)*, **36**.
- [647] Li, Dongjiang, Shi, Xuesong, Long, Qiwei, Liu, Shenghui, Yang, Wei, Wang, Fangshi, Wei, Qi, and Qiao, Fei. 2020. DXSLAM: A Robust and Efficient Visual SLAM System with Deep Features. In: *IEEE/RSJ International conference on intelligent robots and systems (IROS)*.
- [648] Li, Fangting, Zhang, Guoqiang, and Yan, Jun. 2008. Coregistration Based on Sift Algorithm for Synthetic Aperture Radar Interferometry. Pages 123–128 of: *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVII, Part B1.
- [649] Li, H. 2009. Consensus set maximization with guaranteed global optimality for robust geometry estimation. Pages 1074–1080 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [650] Li, Lin, Kong, Xin, Zhao, Xiangrui, Li, Wanlong, Wen, Feng, Zhang, Hongbo, and Liu, Yong. 2021. SA-LOAM: Semantic-aided LiDAR SLAM with Loop Closure. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [651] Li, M., and Mourikis, A. 2013. High-Precision, Consistent EKF-based Visual-Inertial Odometry. *Intl. J. of Robotics Research*, **32**(6), 690–711.
- [652] Li, M., and Mourikis, A. I. 2012 (May 14–18,). Improving the Accuracy of EKF-based Visual-Inertial Odometry. Pages 828–835 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [653] Li, Qing, Chen, Shaoyang, Wang, Cheng, Li, Xin, Wen, Chenglu, Cheng, Ming, and Li, Jonathan. 2019. LO-Net: Deep Real-Time Lidar Odometry. Pages 8465–8474 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [654] Li, Ruihao, Wang, Sen, Long, Zhiqiang, and Gu, Dongbing. 2018. Undeepvo: Monocular visual odometry through unsupervised deep learning. Pages 7286–7291 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [655] Li, Wanhua, Zhou, Renping, Zhou, Jiawei, Song, Yingwei, Herter, Johannes, Qin, Minghan, Huang, Gao, and Pfister, Hanspeter. 2025. 4D LangSplat: 4D Language Gaussian Splatting via Multimodal Large Language Models. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [656] Li, Xingyi, Zhang, Han, and Chen, Weidong. 2023. 4D Radar-based Pose Graph SLAM with Ego-velocity Pre-integration Factor. *IEEE Robotics and Automation Letters*, **8**(8), 5124–5131.
- [657] Li, Xudong, Wang, Zhixiang, Liu, Zihao, Zhang, Yizhai, Zhang, Fan, Yao, Xiuming, and Huang, Panfeng. 2024b. Asynchronous Event-Inertial Odometry using a Unified Gaussian Process Regression Framework. Pages 7773–7778 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [658] Li, Zhengqi, Tucker, Richard, Cole, Forrester, Wang, Qianqian, Jin, Linyi, Ye, Vickie, Kanazawa, Angjoo, Holynski, Aleksander, and Snavely, Noah. 2024c. Megasam: Accurate, fast, and robust structure and motion from casual dynamic videos. *arXiv preprint*.
- [659] Lichtsteiner, Patrick, Posch, Christoph, and Delbruck, Tobi. 2008. A  $128 \times 128$  120 dB  $15\ \mu s$  latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits*, **43**(2), 566–576.
- [660] Lim, Hyungtae, Hwang, Sungwon, and Myung, Hyun. 2021. ERASOR: Ego-centric ratio of pseudo occupancy-based dynamic object removal for static 3D point cloud map building. *IEEE Robotics and Automation Letters*, **6**(2), 2272–2279.
- [661] Lim, Hyungtae, Jang, Seoyeon, Mersch, Benedikt, Behley, Jens, Myung, Hyun, and Stachniss, Cyrill. 2024a. Helimos: A dataset for moving object segmentation in 3d point clouds from heterogeneous lidar sensors. Pages 14087–14094 of: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [662] Lim, Hyungtae, Kim, Beomsoo, Kim, Daebeom, Lee, Eungchang Mason, and Myung, Hyun. 2024b. Quattro++: Robust global registration exploiting ground segmentation for loop closing in LiDAR SLAM. *Intl. J. of Robotics Research*, **43**(5), 685–715.
- [663] Lim, Jaein, and Tsotras, Panagiotis. 2021. A Generalized A\* Algorithm for Finding Globally Optimal Paths in Weighted Colored Graphs. Pages 7503–7509 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [664] Lin, J., and Zhang, F. 2019. Loam\_livox A Robust LiDAR Odometry and

- Mapping LOAM Package for Livox LiDAR. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [665] Lin, Jiarong, and Zhang, Fu. 2020. Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV. Pages 3126–3131 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [666] Lin, Pei-Chun, Komsuoglu, Haldun, and Koditschek, Daniel E. 2005. A leg configuration measurement system for full-body pose estimates in a hexapod robot. *IEEE Trans. Robotics*, **21**(3), 411–422.
- [667] Lin, Tzu-Yuan, Zhang, Ray, Yu, Justin, and Ghaffari, Maani. 2022 (08–11 Nov). Legged Robot State Estimation using Invariant Kalman Filtering and Learned Contact Events. Pages 1057–1066 of: Faust, Aleksandra, Hsu, David, and Neumann, Gerhard (eds), *Conf. on Robot Learning (CoRL)*. Proceedings of Machine Learning Research, vol. 164.
- [668] Lindenberger, Philipp, Sarlin, Paul-Edouard, and Pollefeys, Marc. 2023. Lightglue: Local feature matching at light speed. Pages 17627–17638 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [669] Lipson, Lahav, Teed, Zachary, and Deng, Jia. 2024. Deep patch visual slam. Pages 424–440 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [670] Liso, Lorenzo, Sandström, Erik, Yugay, Vladimir, Van Gool, Luc, and Oswald, Martin R. 2024. Loopy-slam: Dense neural slam with loop closures. Pages 20363–20373 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [671] Liston, Ronald A. 1967. Walking machine studies. *The Military Engineer*, **59**(388), 101–104.
- [672] Liu, Chenxi, Wang, Wei, Liu, Hairong, and Wang, Jun. 2022. Application of Hawk-Eye Technology to Sports Events. Pages 1–5 of: *Intl. Conf. on Information Technology and Contemporary Sports (TCS)*.
- [673] Liu, Hanxiao, Simonyan, Karen, and Yang, Yiming. 2019a. Darts: Differentiable architecture search. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [674] Liu, Liyang, Fryc, Simon, Wu, Lan, Vu, Thanh Long, Paul, Gavin, and Vidal-Calleja, Teresa. 2021a. Active and interactive mapping with dynamic Gaussian process implicit surfaces for mobile manipulators. *IEEE Robotics and Automation Letters*, **6**(2), 3679–3686.
- [675] Liu, P., Orru, Y., Paxton, C., Shafullah, N.M.M., and Pinto, L. 2024a. OK-Robot: What Really Matters in Integrating Open-Knowledge Models for Robotics. *arXiv preprint*.
- [676] Liu, Risheng, Gao, Jiaxin, Zhang, Jin, Meng, Deyu, and Lin, Zhouchen. 2021b. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *IEEE Trans. Pattern Anal. Machine Intell.*, **44**(12), 10045–10067.
- [677] Liu, Shichen, Li, Tianye, Chen, Weikai, and Li, Hao. 2019b. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [678] Liu, Shilong, Zeng, Zhaoyang, Ren, Tianhe, Li, Feng, Zhang, Hao, Yang, Jie, Jiang, Qing, Li, Chunyuan, Yang, Jianwei, Su, Hang, et al. 2024b. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. Pages 38–55 of: *European Conf. on Computer Vision (ECCV)*. Springer.

- [679] Liu, Wenxin, Caruso, David, Ilg, Eddy, Dong, Jing, Mourikis, Anastasios I., Daniilidis, Kostas, Kumar, Vijay R., and Engel, Jakob J. 2020. TLIO: Tight Learned Inertial Odometry. *IEEE Robotics and Automation Letters*, **5**, 5653–5660.
- [680] Liu, Xiaoye. 2008. Airborne LiDAR for DEM generation: some critical issues. *Progress in Physical Geography: Earth and Environment*, **32**(1), 31–49.
- [681] Liu, Xinghua, Xue, Hanjun, Gao, Xiang, Liu, Han, Chen, Badong, and Ge, Shuzhi Sam. 2023a. Cubic B-Spline-Based Feature Tracking for Visual-Inertial Odometry With Event Camera. *IEEE Trans. Instrum. Meas.*, **72**, 1–15.
- [682] Liu, Zhe, Shi, Dianxi, Li, Ruihao, and Yang, Shaowu. 2023b. ESVIO: Event-Based Stereo Visual-Inertial Odometry. *Sensors*, **23**(4).
- [683] Lobo, Jorge, and Dias, Jorge. 2007. Relative Pose Calibration Between Visual and Inertial Sensors. *Intl. J. of Robotics Research*, **26**(6), 561–575.
- [684] Lochman, Yaroslava, Liepieszov, Kostiantyn, Chen, Jianhui, Perdoch, Michal, Zach, Christopher, and Pritts, James. 2021. Babelcalib: A universal approach to calibrating central cameras. Pages 15253–15262 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [685] Lombardi, Stephen, Simon, Tomas, Saragih, Jason, Schwartz, Gabriel, Lehrmann, Andreas, and Sheikh, Yaser. 2019. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Trans. on Graphics (TOG)*.
- [686] Long, A W, Wolfe, K C, Mashner, M J, and Chirikjian, G S. 2012. The Banana Distribution is Gaussian: A Localization Study with Exponential Coordinates. In: *Robotics: Science and Systems (RSS)*.
- [687] Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. 2015. Fully convolutional networks for semantic segmentation. Pages 3431–3440 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [688] Long, Ran, Rauch, Christian, Zhang, Tianwei, Ivan, Vladimir, and Vijayakumar, Sethu. 2021. Rigidfusion: Robot localisation and mapping in environments with large dynamic rigid objects. *IEEE Robotics and Automation Letters*, **6**(2), 3703–3710.
- [689] Loop, C., Cai, Q., Orts-Escolano, S., and Chou, P. A. 2016. A Closed-Form Bayesian Fusion Equation Using Occupancy Probabilities. Pages 380–388 of: *Intl. Conf. on 3D Vision (3DV)*.
- [690] Looper, Samuel, Rodriguez-Puigvert, Javier, Siegwart, Roland, Cadena, Cesar, and Schmid, Lukas. 2023. 3d vsg: Long-term semantic scene change prediction through 3d variable scene graphs. Pages 8179–8186 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [691] Loper, Matthew, Mahmood, Naureen, Romero, Javier, Pons-Moll, Gerard, and Black, Michael J. 2023. SMPL: A skinned multi-person linear model. Pages 851–866 of: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*.
- [692] Loper, Matthew M, and Black, Michael J. 2014. OpenDR: An approximate differentiable renderer. In: *European Conf. on Computer Vision (ECCV)*.
- [693] Lorensen, William E, and Cline, Harvey E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, **21**(4), 163–169.

- [694] Lourakis, Manolis LA, and Argyros, Antonis A. 2005. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? Pages 1526–1531 of: *Intl. Conf. on Computer Vision (ICCV)*, vol. 2. IEEE.
- [695] Lovegrove, S. J. 2011. *Parametric Dense Visual SLAM*. Ph.D. thesis, Imperial College London.
- [696] Lowe, D.G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Intl. J. of Computer Vision*, **60**(2), 91–110.
- [697] Lu, F., and Milios, E. 1997a. Robot pose estimation in unknown environments by matching 2D range scans. *J. of Intelligent and Robotic Systems*, April, 249:275.
- [698] Lu, Feng, and Milios, Evangelos. 1997b. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, **4**, 333–349.
- [699] Lu, Jingpei, Richter, Florian, and Yip, Michael C. 2023a. Markerless Camera-to-Robot Pose Estimation via Self-Supervised Sim-to-Real Transfer. Pages 21296–21306 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [700] Lu, Sha, Xu, Xuecheng, Yin, Huan, Chen, Zexi, Xiong, Rong, and Wang, Yue. 2022. One ring to rule them all: Radon sinogram for place recognition, orientation and translation estimation. Pages 2778–2785 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [701] Lu, Shiyang, Chang, Haonan, Jing, Eric Pu, Bouali, Abdeslam, and Bekris, Kostas. 2023b. OVIR-3D: Open-Vocabulary 3D Instance Retrieval Without Training on 3D Data. In: *Conf. on Robot Learning (CoRL)*.
- [702] Lu, Ziqi, Huang, Qiangqiang, Doherty, Kevin, and Leonard, John J. 2021. Consensus-Informed Optimization Over Mixtures for Ambiguity-Aware Object SLAM. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [703] Lu, Ziqi, Ye, Jianbo, and Leonard, John. 2025. 3DGS-CD: 3D Gaussian Splatting-Based Change Detection for Physical Object Rearrangement. *IEEE Robotics and Automation Letters*.
- [704] Lucas, Bruce D, and Kanade, Takeo. 1981. An iterative image registration technique with an application to stereo vision. Pages 674–679 of: *Intl. Joint Conf. on AI (IJCAI)*, vol. 2.
- [705] Luo, Bin, and Hancock, Edwin R. 1999. Procrustes Alignment with the EM Algorithm. Pages 623–631 of: *Computer Analysis of Images and Patterns, CAIP*. Lecture Notes in Computer Science, vol. 1689. Springer.
- [706] Lupton, T., and Sukkarieh, S. 2012. Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions. *IEEE Trans. Robotics*, **28**(1), 61–76.
- [707] Lusk, Parker C., and How, Jonathan P. 2022. Global Data Association for SLAM with 3D Grassmannian Manifold Objects. Pages 4463–4470 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [708] Lusk, Parker C., Fathian, Kaveh, and How, Jonathan P. 2021a (May). CLIP-PER: A Graph-Theoretic Framework for Robust Data Association. Pages 13828–13834 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [709] Lusk, Parker C., Roy, Ronak, Fathian, Kaveh, and How, Jonathan P. 2021b (Nov.). *MIXER: A Principled Framework for Multimodal, Multiway Data Association*.

- [710] Lusk, Parker C., Parikh, Devarth, and How, Jonathan P. 2023. GraffMatch: Global Matching of 3D Lines and Planes for Wide Baseline LiDAR Registration. *IEEE Robotics and Automation Letters*, **8**(2), 632–639.
- [711] Lv, Jiajun, Xu, Jinhong, Hu, Kewei, Liu, Yong, and Zuo, Xingxing. 2020. Targetless Calibration of LiDAR-IMU System Based on Continuous-time Batch Estimation. Pages 9968–9975 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [712] Lv, Jiajun, Hu, Kewei, Xu, Jinhong, Liu, Yong, Ma, Xiushui, and Zuo, Xingxing. 2021. CLINS: Continuous-time trajectory estimation for LiDAR-inertial system. Pages 6657–6663 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [713] Lv, Zhaoyang, Dellaert, Frank, Rehg, James M, and Geiger, Andreas. 2019. Taking a deeper look at the inverse compositional algorithm. Pages 4581–4590 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [714] Lynch, Kevin M., and Park, Frank C. 2017a. *Modern Robotics - Mechanics, Planning, and Control*. USA: Cambridge University Press. Chap. 4, pages 137–152.
- [715] Lynch, Kevin M., and Park, Frank C. 2017b. *Modern Robotics - Mechanics, Planning, and Control*. USA: Cambridge University Press. Chap. 5, pages 171–190.
- [716] Lynch, Kevin M, Marchuk, Nicholas D, and Elwin, Matthew L. 2015. *Embedded Computing and Mechatronics with the PIC32 Microcontroller*. 1 edn. Newness. Chap. 21.
- [717] Ma, Junyi, Zhang, Jun, Xu, Jintao, Ai, Rui, Gu, Weihao, and Chen, Xieyuanli. 2022. Overlaptransformer: An efficient and yaw-angle-invariant transformer network for lidar-based place recognition. *IEEE Robotics and Automation Letters*, **7**(3), 6958–6965.
- [718] Macenski, Steve, Tsai, David, and Feinberg, Max. 2020. Spatio-temporal voxel layer: A view on robot perception for the dynamic world. *Intl. J. of Advanced Robotic Systems*, **17**(2).
- [719] Maggio, D., and Carlone, L. 2025. Bayesian Fields: Task-driven Open-Set Semantic Gaussian Splatting. *arXiv preprint*.
- [720] Maggio, D., Chang, Y., Hughes, N., Trang, M., Griffith, D., Dougherty, C., Cristofalo, E., Schmid, L., and Carlone, L. 2024. Clio: Real-time Task-Driven Open-Set 3D Scene Graphs. *IEEE Robotics and Automation Letters*, **9**(10), 8921–8928.
- [721] Maggio, Dominic, Lim, Hyungtae, and Carlone, Luca. 2025. VGGT-SLAM: Dense RGB SLAM Optimized on the  $SL(4)$  Manifold. *arXiv preprint arXiv:2505.12549*.
- [722] Mahlknecht, Florian, Gehrig, Daniel, Nash, Jeremy, Rockenbauer, Friedrich M., Morrell, Benjamin, Delaune, Jeff, and Scaramuzza, Davide. 2022. Exploring Event Camera-based Odometry for Planetary Robots. *IEEE Robotics and Automation Letters*, **7**(4), 8651–8658.
- [723] Majumdar, Arjun, Ajay, Anurag, Zhang, Xiaohan, Putta, Pranav, Yenamandra, Sriram, Henaff, Mikael, Silwal, Sneha, Mcvay, Paul, Maksymets, Oleksandr, Arnaud, Sergio, et al. 2024. Openeqa: Embodied question answering in the era of foundation models. Pages 16488–16498 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [724] Malcolm, James, Yalamanchili, Pavan, McClanahan, Chris, Venugopalakrishnan, Vishwanath, Patel, Krunal, and Melonakos, John. 2012. ArrayFire: a GPU acceleration platform. Pages 49–56 of: *Modeling and simulation for defense systems and applications VII*, vol. 8403. SPIE.
- [725] Mangelson, J. G., Dominic, D., Eustice, R. M., and Vasudevan, R. 2018. Pairwise Consistent Measurement Set Maximization for Robust Multi-robot Map Merging. Pages 2916–2923 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [726] Maravgakis, Michael, Argiropoulos, Despina-Ekaterini, Piperakis, Stylianos, and Trahanias, Panos. 2023. Probabilistic Contact State Estimation for Legged Robots using Inertial Information. Pages 12163–12169 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [727] Marck, Jan Willem, Mohamoud, Ali, vd Houwen, Eric, and van Heijster, Rob. 2013. Indoor radar SLAM: A radar application for vision and GPS denied environments. Pages 471–474 of: *2013 European Radar Conference*.
- [728] Marquardt, D.W. 1963. An Algorithm for Least-Squares Estimation of Non-linear Parameters. *J. Soc. Indust. Appl. Math.*, **11**(2), 431–441.
- [729] Marr, David. 1983. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman and Company.
- [730] Marschner, Steve, and Shirley, Peter. 2018. *Fundamentals of computer graphics*. CRC Press.
- [731] Martel, J. 2019. *Unconventional Processing with Unconventional Visual Sensing*. Ph.D. thesis, ETH Zurich.
- [732] Martel, J., and Dudek, P. 2016. Vision Chips with In-pixel Processors for High-performance Low-power Embedded Vision Systems. In: *ASR-MOV Workshop, CGO*.
- [733] Martens, W., Poffet, Y., Soria, P. R., Fitch, R., and Sukkarieh, S. 2017. Geometric Priors for Gaussian Process Implicit Surfaces. *IEEE Robotics and Automation Letters*, 373–380.
- [734] Martinelli, A. 2013 (Nov.). Visual-inertial structure from motion: Observability and resolvability. Pages 4235–4242 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [735] MATLAB. 2010. *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc.
- [736] Matsuki, Hidenobu, von Stumberg, Lukas, Usenko, Vladyslav, Stückler, Jörg, and Cremers, Daniel. 2018. Omnidirectional DSO: Direct Sparse Odometry with Fish-eye Cameras. *IEEE Robotics and Automation Letters*, **3**(4), 3693–3700.
- [737] Matsuki, Hidenobu, Murai, Riku, Kelly, Paul HJ, and Davison, Andrew J. 2024. Gaussian splatting slam. Pages 18039–18048 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [738] Matsuki, Hidenobu, Bae, Gwangbin, and Davison, Andrew. 2025. 4DTAM: Non-Rigid Tracking and Mapping via Dynamic Surface Gaussians. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [739] Maybeck, P. 1979. *Stochastic Models, Estimation and Control*. Vol. 1. New York: Academic Press.
- [740] Mayer, Nikolaus, Ilg, Eddy, Hausser, Philip, Fischer, Philipp, Cremers, Daniel, Dosovitskiy, Alexey, and Brox, Thomas. 2016. A large dataset to

- train convolutional networks for disparity, optical flow, and scene flow estimation. Pages 4040–4048 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [741] Mazur, K., Bae, G., and Davison, A. J. 2024. SuperPrimitive: Scene Reconstruction at a Primitive Level. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [742] Mazur, Kirill, Sucar, Edgar, and Davison, Andrew J. 2023. Feature-realistic neural fusion for real-time, open set scene understanding. Pages 8201–8207 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [743] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. 2017a. SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation? In: *Intl. Conf. on Computer Vision (ICCV)*.
- [744] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. 2017b. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [745] McCormac, John, Clark, Ronald, Bloesch, Michael, Davison, Andrew, and Leutenegger, Stefan. 2018. Fusion++: Volumetric object-level slam. Pages 32–41 of: *2018 international conference on 3D vision (3DV)*. IEEE.
- [746] McDonald, J., Kaess, M., Cadena, C., Neira, J., and Leonard, J.J. 2013. Real-time 6-DOF multi-session visual SLAM over large-scale environments. *Robotics and Autonomous Systems*, **61**(10), 1144–1158.
- [747] McGhee, Robert B. 1968. Some finite state aspects of legged locomotion. *Mathematical Biosciences*, **2**(1-2), 67–84.
- [748] McGhee, Robert B., and Iswandhi, Geoffrey I. 1979. Adaptive locomotion of a multilegged robot over rough terrain. *IEEE Trans. on Systems, Man, and Cybernetics*, **9**(4), 176–182.
- [749] Meagher, D. 1980. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. *Technical Report, Image Processing Laboratory, Rensselaer Polytechnic Institute(IPL-TR-80-111)*.
- [750] Medioni, G., Lee, M.-S., , and Tang, C.-K. 2000. *A Computational Framework for Segmentation and Grouping*. Elsevier.
- [751] Mei, C., Sibley, G., Cummins, M., Newman, P., and Reid, I. 2011. RSLAM: A System for Large-Scale Mapping in Constant-Time Using Stereo. *Intl. J. of Computer Vision*, **94**, 198–214.
- [752] Melkumyan, Arman, and Ramos, Fabio Tozeto. 2009. A sparse covariance function for exact Gaussian process inference in large datasets. In: *Intl. Joint Conf. on AI (IJCAI)*.
- [753] Menze, Moritz, and Geiger, Andreas. 2015. Object scene flow for autonomous vehicles. Pages 3061–3070 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [754] Menze, Moritz, Heipke, Christian, and Geiger, Andreas. 2018. Object Scene Flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*.
- [755] Mersch, Benedikt, Chen, Xieyuanli, Vizzo, Ignacio, Nunes, Lucas, Behley, Jens, and Stachniss, Cyrill. 2022. Receding moving object segmentation in 3d lidar data using sparse 4d convolutions. *IEEE Robotics and Automation Letters*, **7**(3), 7503–7510.

- [756] Mescheder, Lars, Oechsle, Michael, Niemeyer, Michael, Nowozin, Sebastian, and Geiger, Andreas. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [757] Messikommer, Nico, Fang, Carter, Gehrig, Mathias, and Scaramuzza, Davide. 2023. Data-driven feature tracking for event cameras. Pages 5642–5651 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [758] Meta. 2025 (Mar.). *Meta Quest 3: Technical Specifications*.
- [759] Mielle, Malcolm, Magnusson, Martin, and Lilienthal, Achim J. 2019 (Sept.). A comparative analysis of radar and lidar sensing for localization and mapping. In: *European Conf. on Mobile Robots (ECMR)*.
- [760] Miki, Takahiro, Lee, Joonho, Hwangbo, Jemin, Wellhausen, Lorenz, Koltun, Vladlen, and Hutter, Marco. 2022. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, **7**(62), eabk2822.
- [761] Mildenhall, Ben, Srinivasan, Pratul P, Tancik, Matthew, Barron, Jonathan T, Ramamoorthi, Ravi, and Ng, Ren. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. Pages 405–421 of: *European Conf. on Computer Vision (ECCV)*.
- [762] Mildenhall, Ben, Srinivasan, Pratul P, Tancik, Matthew, Barron, Jonathan T, Ramamoorthi, Ravi, and Ng, Ren. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, **65**(1), 99–106.
- [763] Milioto, A., and Stachniss, C. 2019. Bonnet: An Open-source Training and Deployment Framework for Semantic Segmentation in Robotics Using CNNs. Pages 7094–7100 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [764] Milioto, A., Behley, J., McCool, C., and Stachniss, C. 2020. LiDAR Panoptic Segmentation for Autonomous Driving. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [765] Milioto, Andres, Vizzo, Ignacio, Behley, Jens, and Stachniss, Cyrill. 2019. Rangenet++: Fast and accurate lidar semantic segmentation. Pages 4213–4220 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [766] Mitrokhin, Anton, Ye, Chengxi, Fermuller, Cornelia, Aloimonos, Yiannis, and Delbruck, Tobi. 2019. EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras. Pages 6105–6112 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [767] Moeyns, Diederik Paul, Corradi, Federico, Li, Chenghan, Bamford, Simeon A., Longinotti, Luca, Voigt, Fabian F., Berry, Stewart, Taverni, Gemma, Helmchen, Fritjof, and Delbruck, Tobi. 2018. A Sensitive Dynamic and Active Pixel Vision Sensor for Color or Neural Imaging Applications. *IEEE Trans. Biomed. Circuits Syst.*, **12**(1), 123–136.
- [768] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. 2002. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In: *Proc. 19<sup>th</sup> AAAI National Conference on AI*.
- [769] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens,

- D., Vogt, A., and Thrun, S. 2008. Junior: The Stanford entry in the Urban Challenge. *J. of Field Robotics*, **25**(9), 569–597.
- [770] Mopidevi, A.N., Harlow, K., and Heckman, C. 2024 (Oct.). RMap: Millimeter-Wave Radar Mapping Through Volumetric Upsampling. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [771] Moravec, Hans, and Elfes, Alberto. 1985. High resolution maps from wide angle sonar. Pages 116–121 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [772] Moreno-Noguer, Francesc, and Porta, Josep M. 2011. Probabilistic simultaneous pose and non-rigid shape recovery. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [773] Mosher, Ralph S. 1969. Exploring the potential of a quadruped. *SAE Transactions*, 836–843.
- [774] Mourikis, Anastasios I., and Roumeliotis, Stergios I. 2007. A multi-state constraint Kalman filter for vision-aided inertial navigation. Pages 3565–3572 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [775] Mueggler, Elias, Rebecq, Henri, Gallego, Guillermo, Delbruck, Tobi, and Scaramuzza, Davide. 2017. The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM. *Intl. J. of Robotics Research*, **36**(2), 142–149.
- [776] Mueggler, Elias, Gallego, Guillermo, Rebecq, Henri, and Scaramuzza, Davide. 2018. Continuous-Time Visual-Inertial Odometry for Event Cameras. *IEEE Trans. Robotics*, **34**(6), 1425–1440.
- [777] Mühlfellner, Peter, Bürki, Mathias, Bosse, Michael, Derendarz, Wojciech, Philippson, Roland, and Furgale, Paul. 2016. Summary maps for lifelong visual localization. *J. of Field Robotics*, **33**(5), 561–590.
- [778] Mullane, John, Adams, Martin D., and Wijesoma, Wijerupage Sardha. 2006. Evidential versus Bayesian estimation for radar map building. Pages 1–8 of: *Intl. Conf. on Control, Automation, Robotics and Vision (ICARCV)*.
- [779] Mullane, John, Jose, Ebi, Adams, Martin D., and Wijesoma, Wijerupage Sardha. 2007. Including probabilistic target detection attributes into map representations. *J. on Robotics and Autonomous Systems (RAS)*, **55**(1), 72–85.
- [780] Mullane, John, Vo, Ba-Ngu, Adams, Martin D., and Vo, Ba-Tuong. 2011. A Random-Finite-Set Approach to Bayesian SLAM. *IEEE Trans. Robotics*, **27**(2), 268–282.
- [781] Müller, Thomas, Evans, Alex, Schied, Christoph, and Keller, Alexander. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. on Graphics (TOG)*, **41**(4), 1–15.
- [782] Mur-Artal, Raul, and Tardós, Juan D. 2017a. Orb-slam2: An open-source slam system for monocular, stereo, and rgbd cameras. *IEEE Trans. Robotics*, **33**(5), 1255–1262.
- [783] Mur-Artal, Raúl, and Tardós, Juan D. 2017b. Visual-inertial monocular SLAM with map reuse. *IEEE Robotics and Automation Letters*, **2**(2), 796–803.
- [784] Mur-Artal, Raul, Montiel, Jose Maria Martinez, and Tardos, Juan D. 2015. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Trans. Robotics*, **31**(5), 1147–1163.

- [785] Murai, R., Ortiz, J., Saeedi, S., Kelly, P. H. J., and Davison, A. J. 2023. A robot web for distributed many-device localisation. *IEEE Trans. Robotics*.
- [786] Murai, Riku, Dexheimer, Eric, and Davison, Andrew J. 2025. MASt3R-SLAM: Real-Time Dense SLAM with 3D Reconstruction Priors. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [787] Murray, R.M., Li, Z., and Sastry, S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.
- [788] Museth, Ken. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. In: *ACM SIGGRAPH 2021 Talks*. SIGGRAPH '21. ACM.
- [789] Museth, Ken, Lait, Jeff, Johanson, John, Budsberg, Jeff, Henderson, Ron, Alden, Mihai, Cucka, Peter, Hill, David, and Pearce, Andrew. 2013. Open-VDB: an open-source data structure and toolkit for high-resolution volumes. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. ACM.
- [790] Nabarro, S., van der Wilk, M., and Davison, A. J. 2024. Learning in Deep Factor Graphs with Gaussian Belief Propagation. In: *Intl. Conf. on Machine Learning (ICML)*.
- [791] Nam, Hyunwoo, Xu, Qing, and Hong, Dennis. 2020. A Reliable Low-Cost Foot Contact Sensor for Legged Robots. Pages 219–224 of: *Intl. Conf. on Ubiquitous Robots (UR)*.
- [792] Nardi, L., Bodin, B., Zia, M. Z., Mawer, J., Nisbet, A., Kelly, P. H.J., Davison, A. J., Lujan, M., OBoyle, M. F.P., Riley, G., Topham, N., and Furber, S. 2015. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [793] Nardi, L., Bodin, B., Saeedi, S., Vespa, E., Davison, A. J., and Kelly, P. H. J. 2017. Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper. *arXiv preprint*.
- [794] Neira, J., and Tardos, J.D. 2001. Data association in stochastic mapping using the joint compatibility test. *IEEE Trans. Robotics*, **17**(6), 890–897.
- [795] Nematollahi, Iman, DeMoss, Branton, Chandra, Akshay L, Hawes, Nick, Burgard, Wolfram, and Posner, Ingmar. 2025. LUMOS: Language-Conditioned Imitation Learning with World Models. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [796] Newcombe, R. A. 2012. *Dense Visual SLAM*. Ph.D. thesis, Imperial College London.
- [797] Newcombe, Richard A, Lovegrove, Steven J, and Davison, Andrew J. 2011a. DTAM: Dense tracking and mapping in real-time. Pages 2320–2327 of: *Intl. Conf. on Computer Vision (ICCV)*. IEEE.
- [798] Newcombe, Richard A, Izadi, Shahram, Hilliges, Otmar, Molyneaux, David, Kim, David, Davison, Andrew J, Kohi, Pushmeet, Shotton, Jamie, Hodges, Steve, and Fitzgibbon, Andrew. 2011b. Kinectfusion: Real-time dense surface mapping and tracking. Pages 127–136 of: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. Ieee.
- [799] Newcombe, Richard A, Fox, Dieter, and Seitz, Steven M. 2015. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. Pages 343–352 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [800] Ng, Yin Zhi, Choi, Benjamin, Tan, Robby, and Heng, Lionel. 2021. Continuous-time Radar-inertial Odometry for Automotive Radars. Pages

- 323–330 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [801] Ngo, Dat Tien, Östlund, Jonas, and Fua, Pascal. 2016. Template-based monocular 3D shape recovery using Laplacian meshes. *IEEE Trans. Pattern Anal. Machine Intell.*, **38**(1), 172–187.
  - [802] Nicholson, Lachlan, Milford, Michael, and Sünderhauf, Niko. 2019. Quadric-SLAM: Dual Quadrics From Object Detections as Landmarks in Object-Oriented SLAM. *IEEE Robotics and Automation Letters*, **4**(1), 1–8.
  - [803] Nie, Jiawang. 2014. Optimality conditions and finite convergence of Lasserre’s hierarchy. *Mathematical programming*, **146**(1-2), 97–121.
  - [804] Niemeyer, Michael, Mescheder, Lars, Oechsle, Michael, and Geiger, Andreas. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. Pages 3501 – 3512 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. Piscataway, NJ: IEEE.
  - [805] Nießner, Matthias, Zollhöfer, Michael, Izadi, Shahram, and Stamminger, Marc. 2013. Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. on Graphics (TOG)*, **32**(6), 1–11.
  - [806] Nieto, J., Guivant, H., Nebot, E., and Thrun, S. 2003. Real Time Data Association for FastSLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
  - [807] Nistér, D. 2003. An Efficient Solution to the Five-Point Relative Pose Problem. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
  - [808] Nitzberg, Ramon. 1972. Constant-false-alarm-rate signal processors for several types of interference. *IEEE Trans. Aerosp. Electron. Syst.*, 27–34.
  - [809] Niu, Junkai, Zhong, Sheng, Lu, Xiuyuan, Shen, Shaojie, Gallego, Guillermo, and Zhou, Yi. 2025. ESVO2: Direct Visual-Inertial Odometry with Stereo Event Cameras. *IEEE Trans. Robotics*.
  - [810] Niu, Xiaoji, Wu, Yibin, and Kuang, Jian. 2021. Wheel-INS: A Wheel-Mounted MEMS IMU-Based Dead Reckoning System. *IEEE Transactions on Vehicular Technology*, **70**(10), 9814–9825.
  - [811] Nocedal, Jorge, and Wright, Stephen J. 1999. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag.
  - [812] Nuss, Dominik, Reuter, Stephan, Thom, Markus, Yuan, Ting, Krehl, Günther, Maile, Michael, Gern, Axel, and Dietmayer, Klaus. 2018. A random finite set approach for dynamic occupancy grid maps with real-time application. *Intl. J. of Robotics Research*, **37**(8), 841–866.
  - [813] O’Callaghan, Simon T, and Ramos, Fabio T. 2012. Gaussian process occupancy maps. *Intl. J. of Robotics Research*, **31**(1), 42–62.
  - [814] Ochs, Peter, Dosovitskiy, Alexey, Brox, Thomas, and Pock, Thomas. 2015. On iteratively reweighted algorithms for nonsmooth nonconvex optimization in computer vision. *SIAM Journal on Imaging Sciences*, **8**(1), 331–372.
  - [815] Octo Model Team, Ghosh, Dibya, Walke, Homer, Pertsch, Karl, Black, Kevin, Mees, Oier, Dasari, Sudeep, Hejna, Joey, Xu, Charles, Luo, Jian-lan, Kreiman, Tobias, Tan, You Liang, Chen, Lawrence Yunliang, Sanketi, Pannag, Vuong, Quan, Xiao, Ted, Sadigh, Dorsa, Finn, Chelsea, and Levine, Sergey. 2024. Octo: An Open-Source Generalist Robot Policy. In: *Robotics: Science and Systems (RSS)*.

- [816] Oechsle, Michael, Peng, Songyou, and Geiger, Andreas. 2021. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. Pages 5589–5599 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [817] Ogayar-Anguita, C. J., Lopez-Ruiz, A., Rueda-Ruiz, A. J., and Segura-Sanchez, Rafael J. 2023. Nested spatial data structures for optimal indexing of LiDAR data. *ISPRS J. of Photogrammetry and Remote Sensing (JPRS)*, **195**, 287–297.
- [818] Ok, Kyel, Liu, Katherine, Frey, Kris, How, Jonathan P., and Roy, Nicholas. 2019. Robust Object-based SLAM for High-speed Autonomous Navigation. Pages 669–675 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [819] Oleynikova, Helen, Taylor, Zachary, Fehr, Marius, Siegwart, Roland, and Nieto, Juan. 2017. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. Pages 1366–1373 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [820] Oliphant, Travis E. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.
- [821] Olson, E., Leonard, J., and Teller, S. 2006 (May). Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates. Pages 2262–2269 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [822] Olson, Edwin. 2009. Real-Time Correlative Scan Matching. Pages 4387–4393 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [823] Olson, Edwin. 2015. M3RSM: Many-to-many multi-resolution scan matching. Pages 5815–5821 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [824] Olson, Edwin, and Agarwal, Pratik. 2012 (July). Inference on networks of mixtures for robust robot mapping. In: *Robotics: Science and Systems (RSS)*.
- [825] Olson, Edwin, Strom, Johannes, Morton, Ryan, Richardson, Andrew, Ranganathan, Pradeep, Goeddel, Robert, Bulic, Mihai, Crossman, Jacob, and Marinier, Bob. 2012. Progress toward multi-robot reconnaissance and the MAGIC 2010 competition. *J. of Field Robotics*, **29**(5), 762–792.
- [826] Ong, Dexter, Tao, Yuezhan, Murali, Varun, Spasojevic, Igor, Kumar, Vijay, and Chaudhari, Pratik. 2025. ATLAS Navigator: Active Task-driven LAnguage-embedded Gaussian Splatting. In: *arXiv preprint*.
- [827] Open, NN. 2016. An open source neural networks c++ library. URL: <http://opennn.cimne.com> (2016).
- [828] Oquab, Maxime, Darzet, Timothée, Moutakanni, Théo, Vo, Huy V., Szafraniec, Marc, Khalidov, Vasil, Fernandez, Pierre, HAZIZA, Daniel, Massa, Francisco, El-Nouby, Alaaeldin, Assran, Mido, Ballas, Nicolas, Galuba, Wojciech, Howes, Russell, Huang, Po-Yao, Li, Shang-Wen, Misra, Ishan, Rabbat, Michael, Sharma, Vasu, Synnaeve, Gabriel, Xu, Hu, Jegou, Herve, Mairal, Julien, Labatut, Patrick, Joulin, Armand, and Bojanowski, Piotr. 2024. DINOv2: Learning Robust Visual Features without Supervision. *Trans. on Machine Learning Research (TMLR)*. Featured Certification.
- [829] Ortiz, J., Pupilli, M., Leutenegger, S., and Davison, A. J. 2020. Bundle Adjustment on a Graph Processor. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [830] Ortiz, J., Evans, T., and Davison, A. J. 2021. A visual introduction to Gaussian Belief Propagation. *arXiv preprint*.
- [831] Ortiz, Joseph, Clegg, Alexander, Dong, Jing, Sucar, Edgar, Novotny, David, Zollhoefer, Michael, and Mukadam, Mustafa. 2022. iSDF: Real-time neural signed distance fields for robot perception. In: *Robotics: Science and Systems (RSS)*.
- [832] Ozden, Kemal E., Schindler, Konrad, and Van Gool, Luc. 2010. Multibody Structure-from-Motion in Practice. *IEEE Trans. Pattern Anal. Machine Intell.*, **32**(6), 1134–1141.
- [833] Paek, Dong-Hee, KONG, SEUNG-HYUN, and Wijaya, Kevin Tirta. 2022. K-Radar: 4D Radar Object Detection for Autonomous Driving in Various Weather Conditions. Pages 3819–3829 of: *Advances in Neural Information Processing Systems (NIPS)*, vol. 35.
- [834] Paladini, Marco, Del Bue, Alessio, Stosic, Marko, Dodig, Marija, Xavier, Joao, and Agapito, Lourdes. 2009. Factorization for non-rigid and articulated structure using metric projections. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [835] Palieri, Matteo, Morrell, Benjamin, Thakur, Abhishek, Ebadi, Kamak, Nash, Jeremy, Chatterjee, Arghya, Kanellakis, Christoforos, Carbone, Luca, Guaragnella, Cataldo, and Agha-mohammadi, Ali-akbar. 2021. LOCUS: A Multi-Sensor Lidar-Centric Solution for High-Precision Odometry and 3D Mapping in Real-Time. *IEEE Robotics and Automation Letters*, **6**(1), 421–428.
- [836] Pan, Yue, Xiao, Pengchuan, He, Yujie, Shao, Zhenlei, and Li, Zesong. 2021. MULLS: Versatile LiDAR SLAM via multi-metric linear least square. Pages 11633–11640 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [837] Papalia, A., Fishberg, A., O'Neill, B., How, J., Rosen, D., and Leonard, J. 2024a. Certifiably correct range-aided SLAM. *IEEE Trans. Robotics*.
- [838] Papalia, Alan, Tian, Yulun, Rosen, David M., How, Jonathan P., and Leonard, John J. 2024b. An Overview of the Burer-Monteiro Method for Certifiable Robot Perception. *arXiv preprint*, **abs/2410.00117**.
- [839] Parameshwara, Chethan M, Hari, Gokul, Fermüller, Cornelia, Sanket, Nitin J, and Aloimonos, Yiannis. 2022. DiffPoseNet: Direct differentiable camera pose estimation. Pages 6845–6854 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [840] Parashar, Shaifali, Pizarro, Daniel, and Bartoli, Adrien. 2017. Isometric Non-Rigid Shape-from-Motion with Riemannian Geometry Solved in Linear Time. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(10), 2442–2454.
- [841] Parashar, Shaifali, Pizarro, Daniel, and Bartoli, Adrien. 2021. Robust isometric non-rigid structure-from-motion. *IEEE Trans. Pattern Anal. Machine Intell.*, **44**(10), 6409–6423.
- [842] Paredes-Vallés, Federico, Hagenaars, Jesse, Dupeyroux, Julien, Stroobants, Stein, Xu, Yingfu, and de Croon, Guido C. H. E. 2024. Fully neuromorphic vision and control for autonomous drone flight. *Science Robotics*, **9**(90), eadi0591.
- [843] Park, C., Moghadam, P., Kim, S., Elfes, A., Fookes, C., and Sridharan, S. 2018. Elastic LiDAR Fusion: Dense Map-Centric Continuous-Time SLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [844] Park, Jeong Joon, Florence, Peter, Straub, Julian, Newcombe, Richard, and Lovegrove, Steven. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [845] Park, Yeong Sang, Shin, Young-Sik, and Kim, Ayoung. 2020. PhaRaO: Direct Radar Odometry using Phase Correlation. Pages 2617–2623 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [846] Paskin, M.A. 2003. Thin Junction Tree Filters for Simultaneous Localization and Mapping. In: *Intl. Joint Conf. on AI (IJCAI)*.
- [847] Paszke, Adam, Gross, Sam, Massa, Francisco, Lerer, Adam, Bradbury, James, Chanan, Gregory, Killeen, Trevor, Lin, Zeming, Gimelshein, Natalia, Antiga, Luca, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, **32**.
- [848] Patel, Maithili, and Chernova, Sonia. 2022. Proactive robot assistance via spatio-temporal object modeling. *Conf. on Robot Learning (CoRL)*.
- [849] Paton, Michael, MacTavish, Kirk, Warren, Michael, and Barfoot, Timothy D. 2016. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. Pages 1918–1925 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [850] Pattabiraman, Bharath, Patwary, Md. Mostofa Ali, Gebremedhin, Assefaw H., keng Liao, Wei, and Choudhary, Alok. 2015. Fast Algorithms for the Maximum Clique Problem on Massive Graphs with Applications to Overlapping Community Detection. *Internet Mathematics*, **11**(4-5), 421–448.
- [851] Pavlakos, Georgios, Zhou, Xiaowei, Chan, Aaron, Derpanis, Konstantinos G, and Daniilidis, Kostas. 2017. 6-DoF Object Pose from Semantic Keypoints. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [852] Pavlakos, Georgios, Zhu, Luyang, Zhou, Xiaowei, and Daniilidis, Kostas. 2018. Learning to Estimate 3D Human Pose and Shape from a Single Color Image. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [853] Pearl, J. 2017. *Theoretical Impediments to Machine Learning, With Seven Sparks from the Causal Revolution*. Tech. rept. University of California, Los Angeles. Technical Report R-275.
- [854] Pearlmutter, Barak A, and Siskind, Jeffrey Mark. 2008. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, **30**(2), 1–36.
- [855] Peng, Guohao, Li, Heshan, Zhao, Yangyang, Zhang, Jun, Wu, Zhenyu, Zheng, Pengyu, and Wang, Danwei. 2024a (6). TransLoc4D: Transformer-based 4D Radar Place Recognition. Pages 17595–17605 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [856] Peng, Liangzu, Fazlyab, Mahyar, and Vidal, René. 2022. *Towards Understanding The Semidefinite Relaxations of Truncated Least-Squares in Robust Rotation Search*.
- [857] Peng, Liangzu, Kümmeler, Christian, and Vidal, René. 2023a. On the Convergence of IRLS and Its Variants in Outlier-Robust Estimation. Pages 17808–17818 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [858] Peng, Shihan, Zhou, Hanyu, Dong, Hao, Shi, Zhiwei, Liu, Haoyue, Duan, Yuxing, Chang, Yi, and Yan, Luxin. 2024b. CoSEC: A Coaxial Stereo Event Camera Dataset for Autonomous Driving. *arXiv preprint*.
- [859] Peng, Songyou, Niemeyer, Michael, Mescheder, Lars, Pollefeys, Marc, and Geiger, Andreas. 2020. Convolutional occupancy networks. In: *European Conf. on Computer Vision (ECCV)*.
- [860] Peng, Songyou, Genova, Kyle, Jiang, Chiyu, Tagliasacchi, Andrea, Pollefeys, Marc, Funkhouser, Thomas, et al. 2023b. Openscene: 3d scene understanding with open vocabularies. Pages 815–824 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [861] Peng, Xiongfeng, Liu, Zhihua, Li, Weiming, Tan, Ping, Cho, SoonYong, and Wang, Qiang. 2023c. DVI-SLAM: A dual visual inertial SLAM network. *arXiv preprint*.
- [862] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2024c (May). Quantized Visual-Inertial Odometry. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [863] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2024d (May). Ultra-fast Square-Root Filter-based VINS. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [864] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2025 (May). QVIO2: Quantized MAP-based Visual-Inertial Odometry. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [865] Peng, Zhixi, Shao, Tianjia, Yong, Liu, Zhou, Jingke, Yang, Yin, Wang, Jingdong, and Zhou, Kun. 2024e. RTG-SLAM: Real-time 3D Reconstruction at Scale using Gaussian Splatting. *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- [866] Perera, Samunda, and Barnes, Nick. 2012. Maximal cliques based rigid body motion segmentation with a RGB-D camera. Pages 120–133 of: *Asian Conf. on Computer Vision (ACCV)*. Springer.
- [867] Pfister, H., Zwickler, M., v. Baar, J., and Gross, M. 2000. Surfels: surface elements as rendering primitives. In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- [868] Pfreundschuh, Patrick, Hendrikx, Hubertus FC, Reijgwart, Victor, Dubé, Renaud, Siegwart, Roland, and Cramariuc, Andrei. 2021. Dynamic object aware lidar slam based on automatic generation of training data. Pages 11641–11647 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [869] P.H.S. Torr, A. Zisserman. 2000. *MLESAC: A New Robust Estimator with Application to Estimating Image Geometry*. Tech. rept. MSR-TR-99-60. MSR.
- [870] Pineda, Luis, Fan, Taosha, Monge, Maurizio, Venkataraman, Shobha, Sodhi, Paloma, Chen, Ricky TQ, Ortiz, Joseph, DeTone, Daniel, Wang, Austin, Anderson, Stuart, et al. 2022. Theseus: A library for differentiable nonlinear optimization. *Conf. Neural Information Processing Systems (NIPS)*, **35**, 3801–3818.
- [871] Piperakis, Stylianos, and Trahanias, Panos E. 2016. Non-linear ZMP based state estimation for humanoid robot locomotion. Pages 202–209 of: *IEEE Intl. Conf. on Humanoid Robots*.

- [872] Pizzoli, Matia, Forster, Christian, and Scaramuzza, Davide. 2014. RE-MODE: Probabilistic, monocular dense reconstruction in real time. Pages 2609–2616 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [873] Placed, J.A., Strader, J., Carrillo, H., Atanasov, N., Indelman, V., Carlone, L., and Castellanos, J.A. 2023. A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers. *IEEE Trans. Robotics*, **39**(3), 1686–1705.
- [874] Placed, Julio A, and Castellanos, José A. 2022. A general relationship between optimality criteria and connectivity indices for active graph-SLAM. *IEEE Robotics and Automation Letters*, **8**(2), 816–823.
- [875] Plastria, Frank. 2011. *The Weiszfeld Algorithm: Proof, Amendments, and Extensions*. Springer US. Pages 357–389.
- [876] Pock, T. 2008. *Fast Total Variation for Computer Vision*. Ph.D. thesis, Graz University of Technology.
- [877] Pock, T., Grabner, M., and Bischof, H. 2007. Real-time Computation of Variational Methods on Graphics Hardware. In: *Proceedings of the Computer Vision Winter Workshop*.
- [878] Pomerleau, Dean A. 1989. *ALVINN: an autonomous land vehicle in a neural network*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Page 305–313.
- [879] Pomerleau, François, Krüsi, Philipp, Colas, Francis, Furgale, Paul, and Siegwart, Roland. 2014. Long-term 3D map maintenance in dynamic environments. Pages 3712–3719 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [880] Posch, Christoph, Matolin, Daniel, and Wohlgenannt, Rainer. 2011. A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS. *IEEE J. Solid-State Circuits*, **46**(1), 259–275.
- [881] Posch, Christoph, Serrano-Gotarredona, Teresa, Linares-Barranco, Bernabe, and Delbrück, Tobi. 2014. Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output. *Proc. IEEE*, **102**(10), 1470–1484.
- [882] Powell, M.J.D. 1970. A New Algorithm for Unconstrained Optimization. Pages 31–65 of: Rosen, J., Mangasarian, O., and Ritter, K. (eds), *Nonlinear Programming*. Academic Press.
- [883] Premachandra, H. A. G. C., Liu, Ran, Yuen, Chau, and Tan, U-Xuan. 2023. UWB Radar SLAM: An Anchorless Approach in Vision Denied Indoor Environments. *IEEE Robotics and Automation Letters*, **8**(9), 5299–5306.
- [884] Proudman, Alexander, Ramezani, Milad, Digumarti, Sundara Tejaswi, Chebrolu, Nived, and Fallon, Maurice. 2022. Towards real-time forest inventory using handheld LiDAR. *Robotics and Autonomous Systems*, **157**, 104240.
- [885] Pütz, Sebastian, Wiemann, Thomas, Kleine Piening, Malte, and Hertzberg, Joachim. 2021. Continuous Shortest Path Vector Field Navigation on 3D Triangular Meshes for Mobile Robots. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [886] Qadri, M., Manchester, Z., and Kaess, M. 2023. *Learning Covariances for Estimation with Constrained Bilevel Optimization*.
- [887] Qi, Charles R, Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation.

- Pages 652–660 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [888] Qian, Jingxing, Chatrath, Veronica, Yang, Jun, Servos, James, Schoellig, Angela P, and Waslander, Steven L. 2022. Pocd: Probabilistic object-level change detection and volumetric mapping in semi-static scenes. *Robotics: Science and Systems (RSS)*.
- [889] Qian, Jingxing, Chatrath, Veronica, Servos, James, Mavrinac, Aaron, Burgard, Wolfram, Waslander, Steven L, and Schoellig, Angela P. 2023. Povslam: Probabilistic object-aware variational slam in semi-static environments. *Robotics: Science and Systems (RSS)*.
- [890] Qian, Long, Wu, Jie Ying, DiMaio, Simon P, Navab, Nassir, and Kazanzides, Peter. 2019. A review of augmented reality in robotic-assisted surgery. *IEEE Transactions on Medical Robotics and Bionics*, **2**(1), 1–16.
- [891] Qin, Chao, Ye, Haoyang, Pranata, Christian E, Han, Jun, Zhang, Shuyang, and Liu, Ming. 2020. LINS: A Lidar-Inertial State Estimator for Robust and Efficient Navigation. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [892] Qin, Minghan, Li, Wanhua, Zhou, Jiawei, Wang, Haoqian, and Pfister, Hanspeter. 2024. Langsplat: 3d language gaussian splatting. Pages 20051–20060 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [893] Qin, Tong, Li, Peiliang, and Shen, Shaojie. 2018. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Trans. Robotics*, **34**(4), 1004–1020.
- [894] Qiu, Yuheng, Wang, Chen, Wang, Wenshan, Henein, Mina, and Scherer, Sebastian. 2022. AirDOS: Dynamic SLAM benefits from articulated objects. Pages 8047–8053 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [895] Qiu, Yuheng, Wang, Chen, Xu, Can, Chen, Yutian, Zhou, Xunfei, Xia, Youjie, and Scherer, Sebastian. 2024 (May). *AirIMU: Learning Uncertainty Propagation for Inertial Odometry*.
- [896] Radford, Alec, Kim, Jong Wook, Hallacy, Chris, Ramesh, Aditya, Goh, Gabriel, Agarwal, Sandhini, Sastry, Girish, Askell, Amanda, Mishkin, Pamela, Clark, Jack, et al. 2021. Learning transferable visual models from natural language supervision. Pages 8748–8763 of: *Intl. Conf. on Machine Learning (ICML)*. PMLR.
- [897] Radke, Richard J, Andra, Srinivas, Al-Kofahi, Omar, and Roysam, Badri-nath. 2005. Image change detection algorithms: a systematic survey. *IEEE Trans. on Image Processing*, **14**(3), 294–307.
- [898] Rahimi, Ali, and Recht, Benjamin. 2007. Random features for large-scale kernel machines. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 20.
- [899] Raibert, Marc H. 1986. *Legged robots that balance*. MIT press.
- [900] Ramezani, Milad, Khosoussi, Kasra, Catt, Gavin, Moghadam, Peyman, Williams, Jason, Borges, Paulo, Pauling, Fred, and Kottege, Navinda. 2022. Wildcat: Online continuous-time 3d lidar-inertial slam. *arXiv preprint*.
- [901] Ramos, Fabio, and Ott, Lionel. 2016. Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *Intl. J. of Robotics Research*, **35**(14), 1717–1730.

- [902] Rana, Krishan, Haviland, Jesse, Garg, Sourav, Abou-Chakra, Jad, Reid, Ian, and Suenderhauf, Niko. 2023. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning. Pages 23–72 of: *Conf. on Robot Learning (CoRL)*.
- [903] Ranganathan, A., Kaess, M., and Dellaert, F. 2007. Loopy SAM. In: *Intl. Joint Conf. on AI (IJCAI)*.
- [904] Ranjan, Anurag, and Black, Michael J. 2017. Optical flow estimation using a spatial pyramid network. Pages 4161–4170 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [905] Rapp, Matthias, Dietmayer, Klaus, Hahn, Markus, Schuster, Frank, Lombarcher, Jakob, and Dickmann, Jürgen. 2016. FSCD and BASD: Robust landmark detection and description on radar-based grids. Pages 1–4 of: *2016 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*.
- [906] Rasmussen, Carl Edward, and Williams, Christopher KI. 2006. *Gaussian Processes for Machine Learning*. Cambridge, Mass.: MIT Press.
- [907] Ratliff, Nathan D., Silver, David, and Bagnell, J. Andrew. 2009. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, **27**, 25–53.
- [908] Ravi, Nikhila, Gabeur, Valentin, Hu, Yuan-Ting, Hu, Ronghang, Ryali, Chaitanya, Ma, Tengyu, Khedr, Haitham, Rädle, Roman, Rolland, Chloe, Gustafson, Laura, et al. 2024. Sam 2: Segment anything in images and videos. *arXiv preprint*.
- [909] Ravichandran, Z., Peng, L., Hughes, N., Griffith, J.D., and Carlone, L. 2022. Hierarchical Representations and Explicit Memory: Learning Effective Navigation Policies on 3D Scene Graphs using Graph Neural Networks. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. .
- [910] Ray, Aaron, Bradley, Christopher, Carlone, Luca, and Roy, Nicholas. 2024. Task and Motion Planning in Hierarchical 3D Scene Graphs. In: *Intl. Symp. of Robotics Research (ISRR)*.
- [911] Rebecq, Henri, Horstschäfer, Timo, Gallego, Guillermo, and Scaramuzza, Davide. 2017a. EVO: A Geometric Approach to Event-based 6-DOF Parallel Tracking and Mapping in Real-Time. *IEEE Robotics and Automation Letters*, **2**(2), 593–600.
- [912] Rebecq, Henri, Horstschaefer, Timo, and Scaramuzza, Davide. 2017b. Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Nonlinear Optimization. In: *British Machine Vision Conf. (BMVC)*.
- [913] Rebecq, Henri, Gallego, Guillermo, Mueggler, Elias, and Scaramuzza, Davide. 2018a. EMVS: Event-based Multi-View Stereo—3D Reconstruction with an Event Camera in Real-Time. *Intl. J. of Computer Vision*, **126**(12), 1394–1414.
- [914] Rebecq, Henri, Gehrig, Daniel, and Scaramuzza, Davide. 2018b. ESIM: an Open Event Camera Simulator. Pages 969–982 of: *Conf. on Robot Learning (CoRL)*. Proc. Machine Learning Research, vol. 87. PMLR.
- [915] Rebecq, Henri, Ranftl, René, Koltun, Vladlen, and Scaramuzza, Davide. 2021. High Speed and High Dynamic Range Video with an Event Camera. *IEEE Trans. Pattern Anal. Machine Intell.*, **43**(6), 1964–1980.
- [916] Redmon, Joseph, Divvala, Santosh, Girshick, Ross, and Farhadi, Ali. 2015. *You Only Look Once: Unified, Real-Time Object Detection*.

- [917] Redmon, Joseph, Divvala, Santosh, Girshick, Ross, and Farhadi, Ali. 2016. You Only Look Once: Unified, Real-Time Object Detection. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [918] Reed, Scott, Zolna, Konrad, Parisotto, Emilio, Colmenarejo, Sergio Gómez, Novikov, Alexander, Barth-maron, Gabriel, Giménez, Mai, Sulsky, Yury, Kay, Jackie, Springenberg, Jost Tobias, Eccles, Tom, Bruce, Jake, Razavi, Ali, Edwards, Ashley, Heess, Nicolas, Chen, Yutian, Hadsell, Raia, Vinyals, Oriol, Bordbar, Mahyar, and de Freitas, Nando. 2022. A Generalist Agent. *Trans. on Machine Learning Research (TMLR)*. Featured Certification, Outstanding Certification.
- [919] Reijgwart, V., Millane, A., Oleynikova, H., Siegwart, R., Cadena, C., and Nieto, J. 2020. Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps. *IEEE Robotics and Automation Letters*.
- [920] Reijgwart, Victor, Cadena, Cesar, Siegwart, Roland, and Ott, Lionel. 2023-07. Efficient volumetric mapping of multi-scale environments using wavelet-based compression. In: *Robotics: Science and Systems (RSS)*.
- [921] Reimers, Nils, and Gurevych, Iryna. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- [922] Reinbacher, Christian, Munda, Gottfried, and Pock, Thomas. 2017. Real-Time Panoramic Tracking for Event Cameras. Pages 1–9 of: *IEEE Int. Conf. Comput. Photography (ICCP)*.
- [923] Reinstein, Michal, and Hoffmann, Matej. 2011. Dead reckoning in a dynamic quadruped robot: Inertial navigation system aided by a legged odometer. Pages 617–624 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [924] Ren, Yifei, Xu, Binbin, Choi, Christopher L, and Leutenegger, Stefan. 2022. Visual-inertial multi-instance dynamic SLAM with object-level relocalisation. Pages 11055–11062 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [925] Revels, Jarrett, Lubin, Miles, and Papamarkou, Theodore. 2016. Forward-mode automatic differentiation in Julia. *arXiv preprint*.
- [926] Riegler, Gernot, Osman Ulusoy, Ali, and Geiger, Andreas. 2017. Octnet: Learning deep 3d representations at high resolutions. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [927] Robbins, Herbert, and Monro, Sutton. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.
- [928] Rockwell, Chris, Tung, Joseph, Lin, Tsung-Yi, Liu, Ming-Yu, Fouhey, David F, and Lin, Chen-Hsuan. 2025. Dynamic Camera Poses and Where to Find Them. Pages 12444–12455 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [929] Rodrigues, Rômulo T., Tsiogkas, Nikolaos, Pascoal, António, and Aguiar, A. Pedro. 2021. Online Range-Based SLAM Using B-Spline Surfaces. *IEEE Robotics and Automation Letters*, **6**(2), 1958–1965.
- [930] Rodríguez, Juan J. Gómez, Montiel, J.M.M., and Tardós, Juan D. 2024. NR-SLAM: Nonrigid Monocular SLAM. *IEEE Trans. Robotics*, **40**, 4252–4264.
- [931] Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. 2015. U-net: Convolutional networks for biomedical image segmentation. Pages 234–241 of:

- Intl. Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI).* Springer.
- [932] Roriz, Ricardo, Cabral, Jorge, and Gomes, Tiago. 2021. Automotive LiDAR technology: A survey. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, **23**(7), 6282–6297.
- [933] Rosen, David M., Mason, Julian, and Leonard, John J. 2016a. Towards lifelong feature-based mapping in semi-static environments. Pages 1063–1070 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [934] Rosen, David M., Carlone, Luca, Bandeira, Afonso S., and Leonard, John J. 2019. SE-Sync: A Certifiably Correct Algorithm for Synchronization over the Special Euclidean Group. *Intl. J. of Robotics Research*, **38**(2–3), 95–125.
- [935] Rosen, David M., Doherty, Kevin J., Terán Espinoza, Antonio, and Leonard, John J. 2021. Advances in Inference and Representation for Simultaneous Localization and Mapping. *Annual Review of Control, Robotics, and Autonomous Systems*, **4**, 215–242.
- [936] Rosen, D.M., Carlone, L., Bandeira, A.S., and Leonard, J.J. 2016b (December). SE-Sync: A Certifiably Correct Algorithm for Synchronization over the Special Euclidean Group. In: *Intl. Workshop on the Algorithmic Foundations of Robotics (WAFR)*.
- [937] Rosinol, A., Leonard, J.J., and Carlone, L. 2023. NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. .
- [938] Rosinol, Antoni, Gupta, Arjun, Abate, Marcus, Shi, Jingnan, and Carlone, Luca. 2020a. 3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans. *Robotics: Science and Systems (RSS)*.
- [939] Rosinol, Antoni, Abate, Marcus, Chang, Yun, and Carlone, Luca. 2020b. Kimera: an open-source library for real-time metric-semantic localization and mapping. Pages 1689–1696 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [940] Rosinol, Antoni, Violette, Andrew, Abate, Marcus, Hughes, Nathan, Chang, Yun, Shi, Jingnan, Gupta, Arjun, and Carlone, Luca. 2021. Kimera: From SLAM to spatial perception with 3D dynamic scene graphs. *Intl. J. of Robotics Research*, **40**(12-14), 1510–1546.
- [941] Rosinol Vidal, Antoni, Rebecq, Henri, Horstschafer, Timo, and Scaramuzza, Davide. 2018. Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High Speed Scenarios. *IEEE Robotics and Automation Letters*, **3**(2), 994–1001.
- [942] Ross, Stephane, Gordon, Geoffrey, and Bagnell, Drew. 2011 (11–13 Apr). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. Pages 627–635 of: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research, vol. 15.
- [943] Rosten, E., and Drummond, T. 2006. Machine learning for high-speed corner detection. In: *European Conf. on Computer Vision (ECCV)*.
- [944] Roston, G.P., and Krotkov, E.P. 1992. Dead Reckoning Navigation For Walking Robots. Pages 607–612 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, vol. 1.
- [945] Rotella, Nicholas, Bloesch, Michael, Righetti, Ludovic, and Schaal, Stefan.

2014. State estimation for a humanoid robot. Pages 952–958 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [946] Rotella, Nicholas, Schaal, Stefan, and Righetti, Ludovic. 2018. Unsupervised Contact Learning for Humanoid Estimation and Control. Pages 411–417 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [947] Rouveure, R., Faure, P., and Monod, M. 2010. Radar-based SLAM without odometric sensor. In: *ROBOTICS2010 : International workshop of Mobile Robotics for environment/agriculture*.
- [948] Rowell, Joseph, Zhang, Lintong, and Fallon, Maurice. 2024. LiSTA: Geometric Object-Based Change Detection in Cluttered Environments. Pages 3632–3638 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [949] Rubino, Cosimo, Crocco, Marco, and Del Bue, Alessio. 2018. 3D Object Localisation from Multi-View Image Detections. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(6), 1281–1294.
- [950] Rublee, Ethan, Rabaud, Vincent, Konolige, Kurt, and Bradski, Gary. 2011. ORB: An efficient alternative to SIFT or SURF. Pages 2564–2571 of: *Intl. Conf. on Computer Vision (ICCV)*. IEEE.
- [951] Rückert, Darius, Franke, Linus, and Stamminger, Marc. 2022. Adop: Approximate differentiable one-pixel point rendering. *ACM Trans. on Graphics (TOG)*, **41**(4), 1–14.
- [952] Rueckauer, Bodo, and Delbrück, Tobi. 2016. Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor. *Front. Neurosci.*, **10**(176).
- [953] Ruiz-Sarmiento, Jose-Raul, Galindo, Cipriano, and Gonzalez-Jimenez, Javier. 2017. Building Multiversal Semantic Maps for Mobile Robot Operation. *Knowledge-Based Systems*, **119**, 257–272.
- [954] Rünz, M., and Agapito, L. 2017. Co-Fusion: Real-time Segmentation, Tracking and Fusion of Multiple Objects. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [955] Runz, Martin, Buffier, Maud, and Agapito, Lourdes. 2018. Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. Pages 10–20 of: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. IEEE.
- [956] Rusinkiewicz, S., and Levoy, M. 2001. Efficient variants of the ICP algorithm. In: *Proc. of Intl. Conf. on 3-D Digital Imaging and Modeling*.
- [957] Rusu, Radu Bogdan, Blodow, Nico, and Beetz, Michael. 2009. Fast point feature histograms (FPFH) for 3D registration. Pages 3212–3217 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [958] Rusu, Radu Bogdan, Bradski, Gary, Thibaux, Romain, and Hsu, John. 2010. Fast 3d recognition and pose using the viewpoint feature histogram. Pages 2155–2162 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [959] Rünz, Martin, and Agapito, Lourdes. 2017. Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. Pages 4471–4478 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [960] Saftescu, Stefan, Gadd, Matthew, De Martini, Daniele, Barnes, Dan, and Newman, Paul. 2020. Kidnapped Radar: Topological Radar Localisation using Rotationally-Invariant Metric Learning. Pages 4358–4364 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [961] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. 2013. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [962] Salti, Samuele, Tombari, Federico, and Di Stefano, Luigi. 2014. SHOT: Unique signatures of histograms for surface and texture description. *Comput. Vis. Image Underst.*, **125**, 251–264.
- [963] Salzmann, Mathieu, and Fua, Pascal. 2011. Linear local models for monocular reconstruction of deformable surfaces. *IEEE Trans. Pattern Anal. Machine Intell.*, **33**(5), 931–944.
- [964] San Segundo, Pablo, and Artieda, Jorge. 2015. A novel clique formulation for the visual feature matching problem. *Appl. Intelligence*, **43**(2), 325–342.
- [965] Sandström, Erik, Li, Yue, Van Gool, Luc, and Oswald, Martin R. 2023. Point-slam: Dense neural point cloud-based slam. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [966] Särkkä, Simo”. 2011. Linear Operators and Stochastic Partial Differential Equations in Gaussian Process Regression. Pages 151–158 of: *International Conference on Artificial Neural Networks and Machine Learning*.
- [967] Sarlin, Paul-Edouard, Cadena, Cesar, Siegwart, Roland, and Dymczyk, Marcin. 2019. From Coarse to Fine: Robust Hierarchical Localization at Large Scale. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [968] Sarlin, Paul-Edouard, DeTone, Daniel, Malisiewicz, Tomasz, and Rabинovich, Andrew. 2020. Superglue: Learning feature matching with graph neural networks. Pages 4938–4947 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [969] Saunderson, J., Parrilo, P.A., and Willsky, A. 2014 (May). Semidefinite relaxations for optimization problems over rotation matrices. In: *IEEE Conf. on Decision and Control (CDC)*.
- [970] Schauer, Johannes, and Nüchter, Andreas. 2018. The people-re-mover—removing dynamic objects from 3-d point cloud data by traversing a voxel occupancy grid. *IEEE Robotics and Automation Letters*, **3**(3), 1679–1686.
- [971] Schmid, Lukas, Delmerico, Jeffrey, Schönberger, Johannes L, Nieto, Juan, Pollefeys, Marc, Siegwart, Roland, and Cadena, Cesar. 2022. Panoptic multi-tsdfs: a flexible representation for online multi-resolution volumetric mapping and long-term dynamic scene consistency. Pages 8018–8024 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [972] Schmid, Lukas, Andersson, Olov, Sulser, Aurelio, Pfreundschuh, Patrick, and Siegwart, Roland. 2023. Dynablox: Real-time detection of diverse dynamic objects in complex environments. *IEEE Robotics and Automation Letters*.
- [973] Schmid, Lukas, Abate, Marcus, Chang, Yun, and Carlone, Luca. 2024. Khronos: A unified approach for spatio-temporal metric-semantic slam in dynamic environments. In: *Robotics: Science and Systems (RSS)*.
- [974] Schmidt, Tanner, Newcombe, Richard A, and Fox, Dieter. 2014. DART: Dense Articulated Real-Time Tracking. Pages 1–9 of: *Robotics: Science and Systems (RSS)*, vol. 2. Berkeley, CA.

- [975] Schonberger, Johannes L., and Frahm, Jan-Michael. 2016. Structure-from-motion revisited. Pages 4104–4113 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [976] Schouten, Girmi, and Steckel, Jan. 2017. RadarSLAM: Biomimetic SLAM using ultra-wideband pulse-echo radar. Pages 1–8 of: *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*.
- [977] Schubert, D., Demmel, N., Usenko, V., Stueckler, J., and Cremers, D. 2018 (September). Direct Sparse Odometry With Rolling Shutter. In: *European Conf. on Computer Vision (ECCV)*.
- [978] Schubert, D., Demmel, N., von Stumberg, L., Usenko, V., and Cremers, D. 2019 (November). Rolling-Shutter Modelling for Visual-Inertial Odometry. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [979] Schumann, Ole, Hahn, Markus, Scheiner, Nicolas, Weishaupt, Fabio, Tilly, Julius F, Dickmann, Jürgen, and Wöhler, Christian. 2021. RadarScenes: A real-world radar point cloud data set for automotive applications. Pages 1–8 of: *Intl. Conf. on Information Fusion (FUSION)*.
- [980] Schuster, Frank, Keller, Christoph Gustav, Rapp, Matthias, Haueis, Martin, and Curio, Cristóbal. 2016. Landmark based radar SLAM using graph optimization. Pages 2559–2564 of: *IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE.
- [981] Scona, Raluca, Jaimez, Mariano, Petillot, Yvan R, Fallon, Maurice, and Cremers, Daniel. 2018. StaticFusion: Background reconstruction for dense rgb-d slam in dynamic environments. Pages 3849–3856 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [982] Segal, Aleksandr, Haehnel, Dirk, and Thrun, Sebastian. 2009. Generalized-icp. Page 435 of: *Robotics: Science and Systems (RSS)*, vol. 2. Seattle, WA.
- [983] Seidenschwarz, Jenny, Zhou, Qunjie, Duisterhof, Bardienus Pieter, Ramanan, Deva, and Leal-Taixé, Laura. 2025. DynOMo: Online Point Tracking by Dynamic Online Monocular Gaussian Reconstruction. *Intl. Conf. on 3D Vision (3DV)*.
- [984] Semini, C, Tsagarakis, N G, Guglielmino, E, Focchi, M, Cannella, F, and Caldwell, D G. 2011. Design of HyQ – a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, **225**(6), 831–849.
- [985] Semini, Claudio, and Wieber, Pierre-Brice. 2020. *Legged Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg. Pages 1–8.
- [986] Sengupta, Agniva, and Bartoli, Adrien. 2024. ToTem NRS f M: Object-Wise Non-rigid Structure-from-Motion with a Topological Template. *Intl. J. of Computer Vision*, **132**(6), 2135–2176.
- [987] Sethi, Ishwar K., and Jain, Ramesh C. 1987. Finding Trajectories of Feature Points in a Monocular Image Sequence. *IEEE Trans. Pattern Anal. Machine Intell.*, **9**(1), 56–73.
- [988] Sethian, James Albert. 1996. Level set methods: Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science. *Cambridge monographs on applied and computational mathematics*, **3**.
- [989] Shaban, Amirreza, Cheng, Ching-An, Hatch, Nathan, and Boots, Byron.

2019. Truncated back-propagation for bilevel optimization. Pages 1723–1732 of: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR.
- [990] Shafullah, Nur Muhammad Mahi, Cui, Zichen Jeff, Altanzaya, Ariuntuya, and Pinto, Lerrel. 2022. Behavior Transformers: Cloning  $k$  modes with one stone. In: *Conf. Neural Information Processing Systems (NIPS)*.
- [991] Shaikowitz, Lorenzo, Ubellacker, Samuel, and Carlone, Luca. 2024. A certifiable algorithm for simultaneous shape estimation and object tracking. *IEEE Robotics and Automation Letters*.
- [992] Shan, Mo, Feng, Qiaojun, and Atanasov, Nikolay. 2020a. OrcVIO: Object residual constrained visual-inertial odometry. Pages 5104–5111 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [993] Shan, Mo, Feng, Qiaojun, Jau, Youyi, and Atanasov, Nikolay. 2021a. EL-LIPSDF: Joint Object Pose and Shape Optimization with a Bi-level Ellipsoid and Signed Distance Function Description. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [994] Shan, Qi, Curless, Brian, Furukawa, Yasutaka, Hernandez, Carlos, and Seitz, Steven M. 2014. Occluding contours for multi-view stereo. Pages 4002–4009 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [995] Shan, Tixiao, and Englot, Brendan. 2018. LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [996] Shan, Tixiao, Englot, Brendan, Meyers, Drew, Wang, Wei, Ratti, Carlo, and Rus, Daniela. 2020b. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. Pages 5135–5142 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [997] Shan, Tixiao, Englot, Brendan, Duarte, Fábio, Ratti, Carlo, and Rus, Daniela. 2021b. Robust place recognition using an imaging lidar. Pages 5469–5475 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [998] Sheeny, Marcel, De Pellegrin, Emanuele, Mukherjee, Saptarshi, Ahramian, Alireza, Wang, Sen, and Wallace, Andrew. 2021. RADIATE: A radar dataset for automotive perception in bad weather. Pages 1–7 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [999] Shen, Chen, O’Brien, James F., and Shewchuk, Jonathan Richard. 2004. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 9.
- [1000] Shi, Guowei, Yao, Chen, Liu, Xin, Zhao, Yuntian, Zhu, Zheng, and Jia, Zhenzhong. 2024. Foot Vision: A Vision-Based Multi-Functional Sensorized Foot for Quadruped Robots. *IEEE Robotics and Automation Letters*, **9**(7), 6720–6727.
- [1001] Shi, J., and Tomasi, C. 1994. Good features to track. Pages 593–600 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1002] Shi, J., Yang, H., and Carlone, L. 2021. ROBIN: a Graph-Theoretic Approach to Reject Outliers in Robust Estimation using Invariants. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. arXiv preprint: 2011.03659, .
- [1003] Shi, Jingnan, Yang, Heng, and Carlone, Luca. 2023a. Optimal and Robust Category-Level Perception: Object Pose and Shape Estimation From 2-D and 3-D Semantic Keypoints. *IEEE Trans. Robotics*, **39**(5), 4131–4151.

- [1004] Shi, Pengcheng, Zhang, Yongjun, and Li, Jiayuan. 2023b. Lidar-based place recognition for autonomous driving: A survey. *arXiv preprint*.
- [1005] Shi, Xuesong, Li, Dongjiang, Zhao, Pengpeng, Tian, Qinbin, Tian, Yuxin, Long, Qiwei, Zhu, Chunhao, Song, Jingwei, Qiao, Fei, Song, Le, et al. 2020. Are we ready for service robots? the openloris-scene datasets for lifelong slam. Pages 3139–3145 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1006] Shiba, Shintaro, Aoki, Yoshimitsu, and Gallego, Guillermo. 2022. Event Collapse in Contrast Maximization Frameworks. *Sensors*, **22**(14), 1–20.
- [1007] Shiba, Shintaro, Klose, Yannick, Aoki, Yoshimitsu, and Gallego, Guillermo. 2024. Secrets of Event-based Optical Flow, Depth, and Ego-Motion by Contrast Maximization. *IEEE Trans. Pattern Anal. Machine Intell.*, **46**(12), 7742–7759.
- [1008] Shor, N.Z. 1987. Quadratic optimization problems. *Izv. Akad. Nauk SSSR Tekhn. Kibernet.*, **1**, 128–139.
- [1009] Siciliano, Bruno, Sciavicco, Lorenzo, Villani, Luigi, and Oriolo, Giuseppe. 2008. *Robotics: Modelling, Planning and Control*. 1st edn. Springer Publishing Company, Incorporated. Chap. 5.
- [1010] Sim, R., Elinas, P., Griffin, M., Shyr, A., and Little, J.J. 2006 (Jun). Design and Analysis of a Framework for Real-time Vision-based SLAM using Rao-Blackwellised Particle Filters. In: *Conf. on Computer and Robot Vision (CRV)*.
- [1011] Singer, A. 2010. Angular synchronization by eigenvectors and semidefinite programming. *Appl. Comput. Harmon. Anal.*, **30**, 20–36.
- [1012] Sitzmann, Vincent, Zollhöfer, Michael, and Wetzstein, Gordon. 2019. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In: *Advances in Neural Information Processing Systems (NIPS)*.
- [1013] Slavcheva, Miroslava, Baust, Maximilian, Cremers, Daniel, and Ilic, Slobodan. 2017. KillingFusion: Non-rigid 3D reconstruction without correspondences. Pages 1386–1395 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1014] Slonim, Noam, and Tishby, Naftali. 1999. Agglomerative Information Bottleneck. Pages 617–623 of: *Conf. Neural Information Processing Systems (NIPS)*. NIPS’99.
- [1015] Smith, R., and Cheeseman, P. 1987. On the representation and estimation of spatial uncertainty. *Intl. J. of Robotics Research*, **5**(4), 56–68.
- [1016] Soatto, Stefano, and Chiuso, Alessandro. 2016. Visual Representations: Defining Properties and Deep Approximations. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [1017] Sodhi, P., Dexheimer, E., Mukadam, M., Anderson, S., and Kaess, M. 2021. LEO: Learning energy-based models in factor graph optimization. In: *Conf. on Robot Learning (CoRL)*.
- [1018] Sola, Joan, Vidal-Calleja, Teresa, Civera, Javier, and Montiel, J. 2012. Impact of Landmark Parametrization on Monocular EKF-SLAM with Points and Lines. *International Journal of Computer Vision*, 05.
- [1019] Sola, Joan, Deray, Jeremie, and Atchuthan, Dinesh. 2018. A micro Lie theory for state estimation in robotics. *arXiv preprint*.

- [1020] Song, Jingwei, Wang, Jun, Zhao, Liang, Huang, Shoudong, and Dissanayake, Gamini. 2018. Mis-slam: Real-time large-scale dense deformable slam system in minimal invasive surgery based on heterogeneous computing. *IEEE Robotics and Automation Letters*, **3**(4), 4068–4075.
- [1021] Song, Seungwon, Lim, Hyungtae, Lee, Alex Junho, and Myung, Hyun. 2022. DynaVINS: A visual-inertial SLAM for dynamic environments. *IEEE Robotics and Automation Letters*, **7**(4), 11523–11530.
- [1022] Sorkine, Olga, and Alexa, Marc. 2007. As-rigid-as-possible surface modeling. Pages 109–116 of: *Proceedings of the fifth Eurographics symposium on Geometry processing*.
- [1023] Speciale, P., Paudel, D. P., Oswald, M. R., Kroeger, T., Gool, L. V., and Pollefeys, M. 2017 (July). Consensus Maximization with Linear Matrix Inequality Constraints. Pages 5048–5056 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1024] Stadie, Bradly, Zhang, Lunjun, and Ba, Jimmy. 2020. Learning intrinsic rewards as a bi-level optimization problem. Pages 111–120 of: *Conf. on Uncertainty in Artificial Intelligence (UAI)*. PMLR.
- [1025] Stathopoulos, Nikolaos, Lindqvist, Björn, Koval, Anton, akbar Aghamohammadi, Ali, and Nikolakopoulos, George. 2024. FRAME: A Modular Framework for Autonomous Map-merging: Advancements in the Field. *IEEE Trans. Field Robotics*, Apr.
- [1026] Steinbruecker, F., Sturm, J., and Cremers, D. 2011. Real-Time Visual Odometry from Dense RGB-D Images. In: *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*.
- [1027] Steinbruecker, F., Kerl, C., Sturm, J., and Cremers, D. 2013. Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1028] Steinbruecker, F., Sturm, J., and Cremers, D. 2014. Volumetric 3D Mapping in Real-Time on a CPU. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1029] Steinke, Tim, Büchner, Martin, Vödisch, Niclas, and Valada, Abhinav. 2025. Collaborative Dynamic 3D Scene Graphs for Open-Vocabulary Urban Scene Understanding. *arXiv preprint*.
- [1030] Stillwell, J. 2008. *Naive Lie Theory*. Springer.
- [1031] Stork, Johannes A., and Stoyanov, Todor. 2020. Ensemble of Sparse Gaussian Process Experts for Implicit Surface Mapping with Streaming Data. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1032] Stoyanov, T., Saarinen, J.P., Andreasson, H., and Lilienthal, A.J. 2013. Normal Distributions Transform Occupancy Map Fusion: Simultaneous Mapping and Tracking in Large Scale Dynamic Environments. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1033] Strader, J., Ray, A., Arkin, J., Peterson, M.B., Chang, Y., Hughes, N., Bradley, C., Jia, Y., Nieto-Granda, C., Fan, C., Carlone, L., How, J., and Roy, N. 2025. Language-Grounded Hierarchical Planning and Execution with Multi-Robot 3D Scene Graphs. In: *Intl. Sym. on Experimental Robotics (ISER)*.
- [1034] Strader, Jared, Hughes, Nathan, Chen, William, Speranzon, Alberto, and Carlone, Luca. 2024. Indoor and Outdoor 3D Scene Graph Generation via

- Language-Enabled Spatial Ontologies. *IEEE Robotics and Automation Letters*, **9**(6), 4886–4893.
- [1035] Strasdat, H., Davison, A. J., Montiel, José M. M., and Konolige, K. 2011. Double window optimisation for constant time visual SLAM. Pages 2352–2359 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [1036] Strasdat, H., Montiel, Jose M. M., and Davison, A. J. 2012. Visual SLAM: Why filter? *Image and Vision Computing*, **30**(2), 65–77.
- [1037] Straub, Julian, Whelan, Thomas, Ma, Lingni, Chen, Yufan, Wijmans, Erik, Green, Simon, Engel, Jakob J, Mur-Artal, Raul, Ren, Carl, Verma, Shobhit, et al. 2019. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint*.
- [1038] Stückler, J., and Behnke, S. 2014. Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking. *J. of Visual Communication and Image Representation*, **25**(1), 137–147.
- [1039] Stueckler, J., and Behnke, S. 2014. Efficient Deformable Registration of Multi-Resolution Surfel Maps for Object Manipulation Skill Transfer. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1040] Stumberg, Lukas von, and Cremers, Daniel. 2022. DM-VIO: Delayed Marginalization Visual-Inertial Odometry. *IEEE Robotics and Automation Letters*, **7**(2), 1408–1415.
- [1041] Sturm, J., Bylow, E., Kahl, F., and Cremers, D. 2013 (September). CopyMe3D: Scanning and Printing Persons in 3D. In: *German Conference on Pattern Recognition (GCPR)*.
- [1042] Sturm, Jürgen, Engelhard, Nikolas, Endres, Felix, Burgard, Wolfram, and Cremers, Daniel. 2012. A benchmark for the evaluation of RGB-D SLAM systems. Pages 573–580 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1043] Stühmer, J., Gumhold, S., and Cremers, D. 2010 (September). Real-Time Dense Geometry from a Handheld Camera. Pages 11–20 of: *Pattern Recognition (Proc. DAGM)*.
- [1044] Sucar, Edgar, Liu, Shikun, Ortiz, Joseph, and Davison, Andrew J. 2021. imap: Implicit mapping and positioning in real-time. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1045] Suleiman, A., Zhang, Z., Carlone, L., Karaman, S., and Sze, V. 2018. Navion: A Fully Integrated Energy-Efficient Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones. In: *IEEE Symposium on VLSI Circuits (VLSI-Circuits)*.
- [1046] Sumner, Robert W, Schmid, Johannes, and Pauly, Mark. 2007. Embedded deformation for shape manipulation. Pages 80–es of: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- [1047] Sun, Cheng, Sun, Min, and Chen, Hwann-Tzong. 2022. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1048] Sun, Deqing, Yang, Xiaodong, Liu, Ming-Yu, and Kautz, Jan. 2018. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. Pages 8934–8943 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1049] Sun, Jiaming, Shen, Zehong, Wang, Yuang, Bao, Hujun, and Zhou, Xiaowei. 2021a. LoFTR: Detector-free local feature matching with transformers.

- Pages 8922–8931 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1050] Sun, L., and Deng, Z. 2020. Certifiably Optimal and Robust Camera Pose Estimation from Points and Lines. *IEEE Access*.
- [1051] Sun, Lisong C., Bhatt, Neel P., Liu, Jonathan C., Fan, Zhiwen, Wang, Zhangyang, Humphreys, Todd E., and Topcu, Ufuk. 2024a. MM3DGS SLAM: Multi-modal 3D Gaussian Splatting for SLAM Using Vision, Depth, and Inertial Measurements. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1052] Sun, Scott, Melamed, Dennis, and Kitani, Kris. 2021b. IDOL: Inertial deep orientation-estimation and localization. Pages 6128–6137 of: *National Conf. on Artificial Intelligence (AAAI)*, vol. 35.
- [1053] Sun, Shuo, Mielle, Malcolm, Lilienthal, Achim J., and Magnusson, Martin. 2024b. 3QFP: Efficient neural implicit surface reconstruction using Tri-Quadtree and Fourier feature Positional encoding. Pages 4036–4044 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1054] Sun, Tao, Hao, Yan, Huang, Shengyu, Savarese, Silvio, Schindler, Konrad, Pollefeys, Marc, and Armeni, Iro. 2025. Nothing stands still: A spatiotemporal benchmark on 3d point cloud registration under large geometric and temporal change. *ISPRS Journal of Photogrammetry and Remote Sensing*, **220**, 799–823.
- [1055] Sünderhauf, N., and Protzel, P. 2012. Towards a robust back-end for pose graph SLAM. Pages 1254–1261 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1056] Sünderhauf, Niko, and Protzel, Peter. 2012. Switchable constraints for robust pose graph SLAM. Pages 1879–1884 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1057] Sünderhauf, Niko, and Protzel, Peter. 2013. Switchable constraints vs. max-mixture models vs. RRR-a comparison of three approaches to robust pose graph SLAM. Pages 5198–5203 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1058] Sünderhauf, Niko, Dayoub, Feras, McMahon, Sean, Eich, Markus, Upcroft, Ben, and Milford, Michael. 2015. SLAM–quo vadis? In Support of Object Oriented and Semantic SLAM. In: *Robotics: Science and Systems Workshop*. <http://ylatif.github.io/movingsensors/>.
- [1059] Sutter, H. 2011. *Welcome to the Jungle*. URL <https://herbsutter.com/welcome-to-the-jungle>.
- [1060] Sutton, R. 2019. *The Bitter Lesson*. URL <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>.
- [1061] Suzuki, Taro, Kitamura, Mitsunori, Amano, Yoshiharu, and Hashizume, Takumi. 2010. 6-DOF Localization for a Mobile Robot Using Outdoor 3D Voxel Maps. Pages 5737–5743 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1062] Takeuchi, Eiji, Elfes, Alberto, and Roberts, Jonathan. 2015. *Localization and Place Recognition Using an Ultra-Wide Band (UWB) Radar*. Cham: Springer International Publishing. Pages 275–288.
- [1063] Takmaz, Ayça, Fedele, Elisabetta, Sumner, Robert W, Pollefeys, Marc, Tombari, Federico, and Engelmann, Francis. 2023. Openmask3d: Open-vocabulary 3d instance segmentation. *arXiv preprint*.

- [1064] Talbot, W, Nubert, J, Tuna, T, Cadena Lerma, C, Duembgen, F, Tordesillas Torres, J, Barfoot, T D, and Hutter, M. 2025. Continuous-Time State Estimation Methods in Robotics: A Survey. *IEEE Transactions on Robotics (T-RO)*.
- [1065] Tang, Chengzhou, and Tan, Ping. 2019. Ba-net: Dense bundle adjustment network. *Intl. Conf. on Learning Representations (ICLR)*.
- [1066] Tang, Tim Yuqing, De Martini, Daniele, Barnes, Dan, and Newman, Paul. 2020. RSL-Net: Localising in Satellite Images From a Radar on the Ground. *IEEE Robotics and Automation Letters*, **5**(2), 1087–1094.
- [1067] Tang, Yijie, Zhang, Jiazhao, Yu, Zhinan, Wang, He, and Xu, Kai. 2023. Mips-fusion: Multi-implicit-submaps for scalable and robust online neural rgb-d reconstruction. *ACM Trans. on Graphics (TOG)*, **42**(6), 1–16.
- [1068] Tanner, Michael, Piniés, Pedro, Paz, Lina María, and Newman, Paul. 2016. What lies behind: Recovering hidden shape in dense mapping. Pages 979–986 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1069] Tao, Yifu, Ángel Muñoz-Bañón, Miguel, Zhang, Lintong, Wang, Jiahao, Fu, Lanke Frank Tarimo, and Fallon, Maurice. 2025. *The Oxford Spires Dataset: Benchmarking Large-Scale LiDAR-Visual Localisation, Reconstruction and Radiance Field Methods*.
- [1070] Tardós, Juan D., Neira, José, Newman, Paul M., and Leonard, John J. 2002. Robust Mapping and Localization in Indoor Environments Using Sonar Data. *Intl. J. of Robotics Research*, **21**(4), 311–330.
- [1071] Taverni, Gemma, Moeys, Diederik Paul, Li, Chenghan, Cavaco, Celso, Mot-snyi, Vasyl, Bello, David San Segundo, and Delbruck, Tobi. 2018. Front and Back Illuminated Dynamic and Active Pixel Vision Sensors Comparison. *IEEE Trans. Circuits Syst. II (TCSII)*, **65**(5), 677–681.
- [1072] Tavish, K. Mac, and Barfoot, T. D. 2015. At all costs: A comparison of robust cost functions for camera correspondence outliers. Pages 62–69 of: *Conf. on Computer and Robot Vision (CRV)*. IEEE.
- [1073] Taylor, Jonathan, Jepson, Allan D, and Kutulakos, Kiriakos N. 2010. Non-rigid structure from locally-rigid motion. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1074] Team, ALOHA 2, Aldaco, Jorge, Armstrong, Travis, Baruch, Robert, Bingham, Jeff, Chan, Sanky, Draper, Kenneth, Dwibedi, Debidatta, Finn, Chelsea, Florence, Pete, Goodrich, Spencer, Gramlich, Wayne, Hage, Torr, Herzog, Alexander, Hoech, Jonathan, Nguyen, Thinh, Storz, Ian, Tabanpour, Baruch, Takayama, Leila, Tompson, Jonathan, Wahid, Ayzaan, Wahrburg, Ted, Xu, Sichun, Yaroshenko, Sergey, Zakra, Kevin, and Zhao, Tony Z. 2024. *ALOHA 2: An Enhanced Low-Cost Hardware for Bimanual Teleoperation*.
- [1075] Team, Gemini, Anil, Rohan, Borgeaud, Sebastian, Alayrac, Jean-Baptiste, Yu, Jiahui, Soricut, Radu, Schalkwyk, Johan, Dai, Andrew M, Hauth, Anja, Millican, Katie, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint*.
- [1076] Tedrake, Russ, and the Drake Development Team. 2019. *Drake: Model-based design and verification for robotics*.
- [1077] Teed, Zachary, and Deng, Jia. 2018. Deepv2d: Video to depth with differentiable structure from motion. *arXiv preprint*.

- [1078] Teed, Zachary, and Deng, Jia. 2020. RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. *arXiv preprint, abs/2003.12039*.
- [1079] Teed, Zachary, and Deng, Jia. 2021a. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems (NIPS)*, **34**, 16558–16569.
- [1080] Teed, Zachary, and Deng, Jia. 2021b. Tangent space backpropagation for 3d transformation groups. Pages 10338–10347 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1081] Teed, Zachary, Lipson, Lahav, and Deng, Jia. 2023. Deep patch visual odometry. *Advances in Neural Information Processing Systems (NIPS)*, **36**, 39033–39051.
- [1082] Teng, Sangli, Mueller, Mark Wilfried, and Sreenath, Koushil. 2021. Legged Robot State Estimation in Slippery Environments Using Invariant Extended Kalman Filter with Velocity Update. Pages 3104–3110 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1083] Thomas, Hugues, Sivapurapu, Mouli, and Zhang, Jian. 2024. Embedding Pose Graph, Enabling 3D Foundation Model Capabilities with a Compact Representation. *arXiv preprint*.
- [1084] Thrun, S., Liu, Y., Koller, D., Ng, A.Y., Ghahramani, Z., and Durrant-Whyte, H. 2004. Simultaneous Localization and Mapping With Sparse Extended Information Filters. *Intl. J. of Robotics Research*, **23**(7-8), 693–716.
- [1085] Thrun, S., Burgard, W., and Fox, D. 2005. *Probabilistic Robotics*. MIT Press Cambridge.
- [1086] Thrun, Sebastian. 2003. Learning occupancy grid maps with forward sensor models. *Autonomous robots*, **15**, 111–127.
- [1087] Thrun, Sebastian, Gutmann, Jens-Steffen, Fox, Dieter, Burgard, Wolfram, and Kuipers, Benjamin. 1998. Integrating Topological And Metric Maps For Mobile Robot Navigation: A Statistical Approach. Pages 989–995 of: *National Conf. on Artificial Intelligence (AAAI)*, vol. 9.
- [1088] Thrun, Sebastian, et al. 2002. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, **1**(1-35), 1.
- [1089] Tian, Y., Chang, Y., Quang, L., Schang, A., Nieto-Granda, C., How, J.P., and Carlone, L. 2023. Resilient and Distributed Multi-Robot Visual SLAM: Datasets, Experiments, and Lessons Learned. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1090] Tian, Yulun, Khosoussi, Kasra, Rosen, David M., and How, Jonathan P. 2021a. Distributed Certifiably Correct Pose-Graph Optimization. *IEEE Trans. Robotics*, **37**(6), 2137–2156.
- [1091] Tian, Yulun, Khosoussi, Kasra, and How, Jonathan P. 2021b. A resource-aware approach to collaborative loop-closure detection with provable performance guarantees. *Intl. J. of Robotics Research*, **40**(10-11), 1212–1233.
- [1092] Tian, Yulun, Chang, Yun, Arias, Fernando Herrera, Nieto-Granda, Carlos, How, Jonathan P., and Carlone, Luca. 2022. Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems. *IEEE Trans. Robotics*, **38**(4).
- [1093] Tipaldi, Gian Diego, and Arras, Kai O. 2010. Flirt-interest regions for 2d range data. Pages 3616–3622 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.

- [1094] Tirado-Garin, J., and Civera, J. 2025. From Correspondences to Pose: Non-minimal Certifiably Optimal Relative Pose without Disambiguation. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1095] Tishby, Naftali, Pereira, Fernando, and Bialek, William. 2001. The Information Bottleneck Method. *Proc. of the Allerton Conference on Communication, Control and Computation*, **49**(07).
- [1096] Titterton, D., and Weston, J. 2005. *Strapdown Inertial Navigation Technology*. second edn. The Institution of Engineering and Technology.
- [1097] Todd, D. J. 1985. *A brief history of walking machines*. Boston, MA: Springer US. Pages 169–177.
- [1098] Tokui, Seiya, Oono, Kenta, Hido, Shohei, and Clayton, Justin. 2015. Chainer: a next-generation open source framework for deep learning. Pages 1–6 of: *Neural Information Processing Systems Workshops*, vol. 5.
- [1099] Tomasi, Carlo, and Kanade, Takeo. 1992. Shape and motion from image streams under orthography: a factorization method. *Intl. J. of Computer Vision*, **9**(2), 137–154.
- [1100] Tombari, Federico, Salti, Samuele, and Di Stefano, Luigi. 2011. A combined texture-shape descriptor for enhanced 3D feature matching. Pages 809–812 of: *Intl. Conf. on Image Processing (ICIP)*. IEEE.
- [1101] Touvron, Hugo, Lavril, Thibaut, Izacard, Gautier, Martinet, Xavier, Lachaux, Marie-Anne, Lacroix, Timothée, Rozière, Baptiste, Goyal, Naman, Hambro, Eric, Azhar, Faisal, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint*.
- [1102] Trawny, N., and Roumeliotis, S.I. 2005. Indirect Kalman Filter for 3D Attitude Estimation. *Mars Lab, Technical Report Number 2005-002, Rev. 57*.
- [1103] Trevor, Alexander J. B., Rogers, John G., and Christensen, Henrik I. 2012. Planar surface SLAM with 3D and 2D sensors. Pages 3041–3048 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1104] Triggs, Bill, McLauchlan, Philip F, Hartley, Richard I, and Fitzgibbon, Andrew W. 2000. Bundle adjustment—a modern synthesis. Pages 298–372 of: *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*. Springer.
- [1105] Tron, R., Rosen, D., and Carlone, L. 2015. On the Inclusion of Determinant Constraints in Lagrangian Duality for 3D SLAM. In: *Robotics: Science and Systems Workshop*.
- [1106] Trulls, E., Jin, Y., Yi, K.M., Mishkin, D., and Matas, J. 2022. *Image matching challenge*. <https://www.kaggle.com/competitions/image-matching-challenge-2022>. Accessed: 2022.
- [1107] Tsardoulias, Emmanouil G, Iliakopoulou, A, Kargakos, Andreas, and Petrou, Loukas. 2016. A review of global path planning methods for occupancy grid maps regardless of obstacle density. *J. of Intelligent and Robotic Systems*, **84**(1), 829–858.
- [1108] Tschernezki, Vadim, Laina, Iro, Larlus, Diane, and Vedaldi, Andrea. 2022. Neural Feature Fusion Fields: 3D Distillation of Self-Supervised 2D Image Representations. In: *Proceedings of the International Conference on 3D Vision (3DV)*.
- [1109] Tseng, Paul. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, **109**, 475–494.

- [1110] Tunstall, Lewis, Von Werra, Leandro, and Wolf, Thomas. 2022. *Natural language processing with transformers.* ” O'Reilly Media, Inc.”.
- [1111] Ummenhofer, Benjamin, Zhou, Huizhong, Uhrig, Jonas, Mayer, Nikolaus, Ilg, Eddy, Dosovitskiy, Alexey, and Brox, Thomas. 2017. Demon: Depth and motion network for learning monocular stereo. Pages 5038–5047 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1112] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demirish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. 2008. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *J. of Field Robotics*, **25**(8), 425–426.
- [1113] Usenko, Vladislav, Engel, Jakob, Stückler, Jörg, and Cremers, Daniel. 2016. Direct visual-inertial odometry with stereo cameras. Pages 1885–1892 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1114] Usenko, Vladislav, Demmel, Nikolaus, and Cremers, Daniel. 2018. The double sphere camera model. Pages 552–560 of: *2018 International Conference on 3D Vision (3DV)*. IEEE.
- [1115] Usenko, Vladislav, Demmel, Nikolaus, Schubert, David, Stückler, Jörg, and Cremers, Daniel. 2019. Visual-inertial mapping with non-linear factor recovery. *IEEE Robotics and Automation Letters*, **5**(2), 422–429.
- [1116] Uy, Mikaela Angelina, and Lee, Gim Hee. 2018. Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. Pages 4470–4479 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1117] van den Oord, Aäron, Li, Yazhe, and Vinyals, Oriol. 2018. Representation Learning with Contrastive Predictive Coding. *arXiv preprint*.
- [1118] Varghese, Rejin, and M., Sambath. 2024. YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. In: *International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*.
- [1119] Vasudevan, Shrihari, Ramos, Fabio, Nettleton, Eric, and Durrant-Whyte, Hugh. 2009. Gaussian process modeling of large-scale terrain. Pages 812–840 of: *J. of Field Robotics*, vol. 26.
- [1120] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Lukasz, and Polosukhin, Illia. 2017. Attention is all you need. *Advances in Neural Information Processing Systems (NIPS)*, **30**.
- [1121] Vespa, E., Nikolov, N., Grimm, M., Nardi, L., Kelly, P. H.J., and Leutenegger, S. 2018. Efficient Octree-based Volumetric SLAM Supporting Signed-distance and Occupancy Mapping. *IEEE Robotics and Automation Letters*, **3**(2), 1144–1151.
- [1122] Vespa, Emanuele, Funk, Nils, Kelly, Paul HJ, and Leutenegger, Stefan. 2019. Adaptive-resolution octree-based volumetric SLAM. Pages 654–662 of: *Intl. Conf. on 3D Vision (3DV)*.
- [1123] Vial, Pau, Solà, Joan, Palomeras, Narcís, and Carreras, Marc. 2024. On Lie

- group IMU and linear velocity preintegration for autonomous navigation considering the Earth rotation compensation. *IEEE Trans. Robotics*, 1–18.
- [1124] Vicente, Sara, and Agapito, Lourdes. 2012. Soft inextensibility constraints for template-free non-rigid reconstruction. In: *European Conf. on Computer Vision (ECCV)*.
- [1125] Vigne, Matthieu, Khouri, Antonio El, Pétriaux, Marine, Meglio, Florent Di, and Petit, Nicolas. 2022. MOVIE: A Velocity-Aided IMU Attitude Estimator for Observing and Controlling Multiple Deformations on Legged Robots. *IEEE Robotics and Automation Letters*, **7**(2), 3969–3976.
- [1126] Vijayanarasimhan, Sudheendra, Ricco, Susanna, Schmid, Cordelia, Sukthankar, Rahul, and Fragkiadaki, Katerina. 2017. Sfm-net: Learning of structure and motion from video. *arXiv preprint*.
- [1127] Virgolino Soares, João Carlos, Medeiros, Vivian Suzano, Abati, Gabriel Fischer, Becker, Marcelo, Caurin, Glauco, Gattass, Marcelo, and Meggiolaro, Marco Antonio. 2023. Visual localization and mapping in dynamic and changing environments. *J. of Intelligent and Robotic Systems*, **109**(4), 95.
- [1128] Vizzo, I., Chen, X., Chebrolu, N., Behley, J., and Stachniss, C. 2021. Poisson Surface Reconstruction for LiDAR Odometry and Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1129] Vizzo, Ignacio, Guadagnino, Tiziano, Behley, Jens, and Stachniss, Cyrill. 2022. VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data. *IEEE Sensors*, **22**(3).
- [1130] Vizzo, Ignacio, Guadagnino, Tiziano, Mersch, Benedikt, Wiesmann, Louis, Behley, Jens, and Stachniss, Cyrill. 2023. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters*, **8**(2), 1029–1036.
- [1131] Vödisch, Niclas, Cattaneo, Daniele, Burgard, Wolfram, and Valada, Abhinav. 2022. Continual slam: Beyond lifelong simultaneous localization and mapping through continual learning. Pages 19–35 of: *Intl. Symp. of Robotics Research (ISRR)*. Springer.
- [1132] von Stumberg, L., and Cremers, D. 2022. DM-VIO: Delayed Marginalization Visual-Inertial Odometry. *IEEE Robotics and Automation Letters*, **7**(2), 1408–1415.
- [1133] Von Stumberg, Lukas, Usenko, Vladislav, and Cremers, Daniel. 2018. Direct sparse visual-inertial odometry using dynamic marginalization. Pages 2510–2517 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1134] Von Stumberg, Lukas, Wenzel, Patrick, Khan, Qadeer, and Cremers, Daniel. 2020a. Gn-net: The gauss-newton loss for multi-weather relocalization. *IEEE Robotics and Automation Letters*, **5**(2), 890–897.
- [1135] Von Stumberg, Lukas, Wenzel, Patrick, Yang, Nan, and Cremers, Daniel. 2020b. Lm-reloc: Levenberg-marquardt based direct visual relocalization. Pages 968–977 of: *Intl. Conf. on 3D Vision (3DV)*. IEEE.
- [1136] Wächter, Andreas, and Biegler, Lorenz T. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, **106**(1), 25–57.
- [1137] Wald, Johanna, Avetisyan, Armen, Navab, Nassir, Tombari, Federico, and Nießner, Matthias. 2019. Rio: 3d object instance re-localization in changing indoor environments. Pages 7658–7667 of: *Intl. Conf. on Computer Vision (ICCV)*.

- [1138] Wald, Johanna, Dhamo, Helisa, Navab, Nassir, and Tombari, Federico. 2020. Learning 3D Semantic Scene Graphs from 3D Indoor Reconstructions. Pages 3961–3970 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1139] Walke, Homer, Black, Kevin, Lee, Abraham, Kim, Moo Jin, Du, Max, Zheng, Chongyi, Zhao, Tony, Hansen-Estruch, Philippe, Vuong, Quan, He, Andre, Myers, Vivek, Fang, Kuan, Finn, Chelsea, and Levine, Sergey. 2023. Bridge-Data V2: A Dataset for Robot Learning at Scale. In: *Conference on Robot Learning (CoRL)*.
- [1140] Wang, Chen, Gao, Dasong, Xu, Kuan, Geng, Junyi, Hu, Yaoyu, Qiu, Yuheng, Li, Bowen, Yang, Fan, Moon, Brady, Pandey, Abhinav, et al. 2023a. *FastTriggs*. <https://pypose.org/docs/main/generated/pypose.optim.corrector.FastTriggs/>.
- [1141] Wang, Chen, Gao, Dasong, Xu, Kuan, Geng, Junyi, Hu, Yaoyu, Qiu, Yuheng, Li, Bowen, Yang, Fan, Moon, Brady, Pandey, Abhinav, et al. 2023b. *LieTensor*. <https://pypose.org/docs/main/generated/pypose.LieTensor>.
- [1142] Wang, Chen, Gao, Dasong, Xu, Kuan, Geng, Junyi, Hu, Yaoyu, Qiu, Yuheng, Li, Bowen, Yang, Fan, Moon, Brady, Pandey, Abhinav, Aryan, Xu, Jiahe, Wu, Tianhao, He, Haonan, Huang, Daning, Ren, Zhongqiang, Zhao, Shibo, Fu, Taimeng, Reddy, Pranay, Lin, Xiao, Wang, Wenshan, Shi, Jingnan, Talak, Rajat, Cao, Kun, Du, Yi, Wang, Han, Yu, Huai, Wang, Shanzhao, Chen, Siyu, Kashyap, Ananth, Bandaru, Rohan, Dantu, Karthik, Wu, Jiajun, Xie, Lihua, Carbone, Luca, Hutter, Marco, and Scherer, Sebastian. 2023c. PyPose: A Library for Robot Learning with Physics-based Optimization. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1143] Wang, Chen, Gao, Dasong, Xu, Kuan, Geng, Junyi, Hu, Yaoyu, Qiu, Yuheng, Li, Bowen, Yang, Fan, Moon, Brady, Pandey, Abhinav, et al. 2023d. *PyPose Documents*. <https://pypose.org/docs/>.
- [1144] Wang, Chen, Gao, Dasong, Xu, Kuan, Geng, Junyi, Hu, Yaoyu, Qiu, Yuheng, Li, Bowen, Yang, Fan, Moon, Brady, Pandey, Abhinav, et al. 2023e. *PyPose Tutorial*. <https://github.com/pypose/tutorials>.
- [1145] Wang, Chen, Ji, Kaiyi, Geng, Junyi, Ren, Zhongqiang, Fu, Taimeng, Yang, Fan, Guo, Yifan, He, Haonan, Chen, Xiangyu, Zhan, Zitong, Du, Qiwei, Su, Shaoshu, Li, Bowen, Qiu, Yuheng, Lin, Xiao, Du, Yi, Li, Qihang, and Zhao, Zhipeng. 2025a. Imperative Learning: A Self-supervised Neural-Symbolic Learning Framework for Robot Autonomy. *Intl. J. of Robotics Research*.
- [1146] Wang, Chieh-Chih, Thorpe, Charles, Thrun, Sebastian, Hebert, Martial, and Durrant-Whyte, Hugh. 2007. Simultaneous Localization, Mapping and Moving Object Tracking. *Intl. J. of Robotics Research*, **26**(9), 889–916.
- [1147] Wang, H., Wang, C., Chen, C., and Xie, L. 2021a. F-LOAM: Fast LiDAR Odometry and Mapping. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1148] Wang, Han, Wang, Chen, and Xie, Lihua. 2020a. Intensity scan context: Coding intensity and geometry relations for loop closure detection. Pages 2095–2101 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1149] Wang, He, Sridhar, Srinath, Huang, Jingwei, Valentin, Julien, Song, Shuran, and Guibas, Leonidas J. 2019a. Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [1150] Wang, Hengyi, Wang, Jingwen, and Agapito, Lourdes. 2023f. Co-slam: Joint coordinate and sparse parametric encodings for neural real-time slam. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1151] Wang, Jianeng, and Gammell, Jonathan D. 2023. Event-Based Stereo Visual Odometry With Native Temporal Resolution via Continuous-Time Gaussian Process Regression. *IEEE Robotics and Automation Letters*, **8**(10), 6707–6714.
- [1152] Wang, Jianyuan, Karaev, Nikita, Rupprecht, Christian, and Novotny, David. 2024a. Vggsfm: Visual geometry grounded deep structure from motion. Pages 21686–21697 of: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.
- [1153] Wang, Jianyuan, Chen, Minghao, Karaev, Nikita, Vedaldi, Andrea, Rupprecht, Christian, and Novotny, David. 2025b. VGGT: Visual Geometry Grounded Transformer. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- [1154] Wang, Jinkun, and Englot, Brendan. 2016. Fast, accurate gaussian process occupancy maps via test-data octrees and nested Bayesian fusion. Pages 1003–1010 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1155] Wang, K., Gao, F., and Shen, S. 2019b. Real-Time Scalable Dense Surfel Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1156] Wang, Peng, Liu, Lingjie, Liu, Yuan, Theobalt, Christian, Komura, Taku, and Wang, Wenping. 2021b. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In: *Advances in Neural Information Processing Systems (NIPS)*.
- [1157] Wang, R., Schwörer, M., and Cremers, D. 2017a (October). Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1158] Wang, Rui, Yang, Nan, Stückler, Jörg, and Cremers, Daniel. 2020b. DirectShape: Photometric Alignment of Shape Priors for Visual Vehicle Pose and Shape Estimation. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1159] Wang, Ruicheng, Xu, Sicheng, Dai, Cassie, Xiang, Jianfeng, Deng, Yu, Tong, Xin, and Yang, Jiaolong. 2025c. Moge: Unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1160] Wang, Sen, Clark, Ronald, Wen, Hongkai, and Trigoni, Niki. 2017b. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. Pages 2043–2050 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1161] Wang, Shuzhe, Leroy, Vincent, Cabon, Yohann, Chidlovskii, Boris, and Revaud, Jerome. 2024b. Dust3r: Geometric 3d vision made easy. Pages 20697–20709 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1162] Wang, T., Dhiman, V., and Atanasov, N. 2023g. Inverse Reinforcement Learning for Autonomous Navigation via Differentiable Semantic Mapping and Planning. *Autonomous Robots*, **47**(6), 809–830.
- [1163] Wang, Wenshan, Zhu, Delong, Wang, Xiangwei, Hu, Yaoyu, Qiu, Yuheng, Wang, Chen, Hu, Yafei, Kapoor, Ashish, and Scherer, Sebastian. 2020c. Tar-tanair: A dataset to push the limits of visual slam. Pages 4909–4916 of:

- 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE.
- [1164] Wang, Wenshan, Hu, Yaoyu, and Scherer, Sebastian. 2021c. Tartanvo: A generalizable learning-based vo. Pages 1761–1772 of: *Conf. on Robot Learning (CoRL)*. PMLR.
- [1165] Wang, Yi Ru, Duan, Jiafei, Fox, Dieter, and Srinivasa, Siddhartha. 2023h. NEWTON: Are Large Language Models Capable of Physical Reasoning? *arXiv preprint*.
- [1166] Warburg, Frederik, Hauberg, Soren, Lopez-Antequera, Manuel, Gargallo, Pau, Kuang, Yubin, and Civera, Javier. 2020. Mapillary street-level sequences: A dataset for lifelong place recognition. Pages 2626–2635 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1167] Weber, S., Demmel, N., and Cremers, D. 2021. Multidirectional Conjugate Gradients for Scalable Bundle Adjustment. In: *German Conf. on Pattern Recognition (GCPR)*.
- [1168] Weber, S., Demmel, N., Chan, T Chon, and Cremers, D. 2023. Power Bundle Adjustment for Large-Scale 3D Reconstruction. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1169] Weber, S., Hong, JH, and Cremers, D. 2024. Power Variable Projection for Initialization-Free Large-Scale Bundle Adjustment. In: *European Conference on Computer Vision (ECCV)*.
- [1170] Wei, Peng, Hua, Guoliang, Huang, Weibo, Meng, Fanyang, and Liu, Hong. 2021. Unsupervised monocular visual-inertial odometry network. Pages 2347–2354 of: *Intl. Joint Conf. on AI (IJCAI)*.
- [1171] Wei, T., Patel, Y., Shekhovtsov, A., Matas, J., and Barath, D. 2023. Generalized differentiable RANSAC. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1172] Weikersdorfer, David, Hoffmann, Raoul, and Conradt, Jörg. 2013. Simultaneous Localization and Mapping for event-based Vision Systems. Pages 133–142 of: *Int. Conf. Comput. Vis. Syst. (ICVS)*.
- [1173] Weikersdorfer, David, Adrian, David B., Cremers, Daniel, and Conradt, Jörg. 2014. Event-based 3D SLAM with a depth-augmented dynamic vision sensor. Pages 359–364 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1174] Weinzaepfel, Philippe, Leroy, Vincent, Lucas, Thomas, Brégier, Romain, Cabon, Yohann, Arora, Vaibhav, Antsfeld, Leonid, Chidlovskii, Boris, Csurka, Gabriela, and Revaud, Jérôme. 2022. Croco: Self-supervised pre-training for 3d vision tasks by cross-view completion. Pages 3502–3516 of: *Conf. Neural Information Processing Systems (NIPS)*, vol. 35.
- [1175] Weitkamp, Claus. 2006. *Lidar: range-resolved optical remote sensing of the atmosphere*. Vol. 102. Springer Verlag.
- [1176] Wen, Bowen, and Bekris, Kostas. 2021. Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models. Pages 8067–8074 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1177] Wendel, Andreas, Maurer, Michael, Graber, Gottfried, Pock, Thomas, and Bischof, Horst. 2012. Dense reconstruction on-the-fly. Pages 1450–1457 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- [1178] Wenzel, P., Wang, R., Yang, N., Cheng, Q., Khan, Q., von Stumberg, L., Zeller, N., and Cremers, D. 2020. 4Seasons: A Cross-Season Dataset for

- Multi-Weather SLAM in Autonomous Driving. In: *German Conf. on Pattern Recognition (GCPR)*.
- [1179] Werby, Abdelrhman, Huang, Chenguang, Büchner, Martin, Valada, Abhinav, and Burgard, Wolfram. 2024. Hierarchical Open-Vocabulary 3D Scene Graphs for Language-Grounded Robot Navigation. *Robotics: Science and Systems (RSS)*.
- [1180] Whelan, Thomas, Kaess, Michael, Fallon, Maurice, Johannsson, Hordur, Leonard, John, and McDonald, John. 2012 (July). Kintinuous: Spatially extended kinectfusion. In: *Robotics: Science and Systems Workshop*.
- [1181] Whelan, Thomas, Leutenegger, Stefan, Salas-Moreno, Renato F, Glocker, Ben, and Davison, Andrew J. 2015. ElasticFusion: Dense SLAM without a pose graph. Page 3 of: *Robotics: Science and Systems (RSS)*, vol. 11. Rome.
- [1182] Williams, Christopher, and Seeger, Matthias. 2000. Using the Nyström method to speed up kernel machines. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 13.
- [1183] Williams, Oliver, and Fitzgibbon, Andrew. 2007. *Gaussian Process Implicit Surfaces*.
- [1184] Wimbauer, Felix, Yang, Nan, Von Stumberg, Lukas, Zeller, Niclas, and Cremers, Daniel. 2021. Monorec: Semi-supervised dense reconstruction in dynamic environments from a single moving camera. Pages 6112–6122 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1185] Wimbauer, Felix, Yang, Nan, Rupprecht, Christian, and Cremers, Daniel. 2023. Behind the scenes: Density fields for single view reconstruction. Pages 9076–9086 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1186] Wimbauer, Felix, Chen, Weirong, Muhle, Dominik, Rupprecht, Christian, and Cremers, Daniel. 2025. AnyCam: Learning to Recover Camera Poses and Intrinsics from Casual Videos. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1187] Wisth, David, Camurri, Marco, and Fallon, Maurice F. 2020. Preintegrated Velocity Bias Estimation to Overcome Contact Nonlinearities in Legged Robot Odometry. Pages 392–398 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1188] Wisth, David, Camurri, Marco, and Fallon, Maurice. 2022. Vilens: Visual, inertial, lidar, and leg odometry for all-terrain legged robots. *IEEE Trans. Robotics*, **39**(1), 309–326.
- [1189] Wu, Guanjun, Yi, Taoran, Fang, Jiemin, Xie, Lingxi, Zhang, Xiaopeng, Wei, Wei, Liu, Wenyu, Tian, Qi, and Wang, Xinggang. 2024a. 4d gaussian splatting for real-time dynamic scene rendering. Pages 20310–20320 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1190] Wu, Hao, Sankaranarayanan, Aswin C., and Chellappa, Rama. 2007. In Situ Evaluation of Tracking Algorithms Using Time Reversed Chains. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1191] Wu, Kejian J, Ahmed, Ahmed M, Georgiou, Georgios A, and Roumeliotis, Stergios I. 2015. A square root inverse filter for efficient vision-aided inertial navigation on mobile devices. In: *Robotics: Science and Systems Conference (RSS)*.
- [1192] Wu, Lan, Lee, Ki Myung Brian, Liu, Liyang, and Vidal-Calleja, Teresa.

- 2021a. Faithful Euclidean distance field from log-Gaussian process implicit surfaces. *IEEE Robotics and Automation Letters*, 2461–2468.
- [1193] Wu, Lan, Lee, Ki Myung Brian, Le Gentil, Cedric, and Vidal-Calleja, Teresa. 2023a. Log-GPIS-MOP: A Unified Representation for Mapping, Odometry, and Planning. *IEEE Trans. Robotics*, **39**(5), 4078–4094.
- [1194] Wu, Qinghua, and Hao, Jin-Kao. 2015. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, **242**(3), 693–709.
- [1195] Wu, Shun-Cheng, Wald, Johanna, Tateno, Keisuke, Navab, Nassir, and Tombari, Federico. 2021b. SceneGraphFusion: Incremental 3D Scene Graph Prediction from RGB-D Sequences. Pages 7515–7525 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1196] Wu, Xiaoyi, Chen, Yushuai, Li, Zhan, Hong, Ziyang, and Hu, Liang. 2024b. EFEAR-4D: Ego-Velocity Filtering for Efficient and Accurate 4D Radar Odometry. *IEEE Robotics and Automation Letters*, **9**(11), 9828–9835.
- [1197] Wu, Yibin, Kuang, Jian, Niu, Xiaoji, Behley, Jens, Klingbeil, Lasse, and Kuhlmann, Heiner. 2023b. Wheel-SLAM: Simultaneous Localization and Terrain Mapping Using One Wheel-Mounted IMU. *IEEE Robotics and Automation Letters*, **8**(1), 280–287.
- [1198] Wu, Yuchen, Yoon, David J., Burnett, Keenan, Kammel, Soeren, Chen, Yi, Vhavle, Heethesh, and Barfoot, Timothy D. 2023c. Picking Up Speed: Continuous-Time Lidar-Only Odometry using Doppler Velocity Measurements. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1199] Xie, Yiheng, Takikawa, Towaki, Saito, Shunsuke, Litany, Or, Yan, Shiqin, Khan, Numair, Tombari, Federico, Tompkin, James, Sitzmann, Vincent, and Sridhar, Srinath. 2022. Neural fields in visual computing and beyond. Pages 641–676 of: *Computer Graphics Forum*, vol. 41. Wiley Online Library.
- [1200] Xinjilefu, X., Feng, Siyuan, Huang, Weiwei, and Atkeson, Christopher G. 2014a. Decoupled state estimation for humanoids using full-body dynamics. Pages 195–201 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1201] Xinjilefu, X., Feng, Siyuan, and Atkeson, Christopher G. 2014b. Dynamic state estimation using Quadratic Programming. Pages 989–994 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1202] Xinjilefu, X, Feng, Siyuan, and Atkeson, Christopher G. 2016. A distributed MEMS gyro network for joint velocity estimation. Pages 1879–1884 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1203] Xu, Binbin, Li, Wenbin, Tzoumanikas, Dimos, Bloesch, Michael, Davison, Andrew, and Leutenegger, Stefan. 2019. Mid-fusion: Octree-based object-level multi-instance dynamic slam. Pages 5231–5237 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1204] Xu, Binbin, Davison, Andrew J, and Leutenegger, Stefan. 2022a. Learning to complete object shapes for object-level mapping in dynamic scenes. Pages 2257–2264 of: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [1205] Xu, Kuan, Hao, Yuefan, Yuan, Shenghai, Wang, Chen, and Xie, Lihua. 2025. AirSLAM: An Efficient and Illumination-Robust Point-Line Visual SLAM System. *IEEE Trans. Robotics*.
- [1206] Xu, Qiangeng, Xu, Zexiang, Philip, Julien, Bi, Sai, Shu, Zhixin, Sunkavalli,

- Kalyan, and Neumann, Ulrich. 2022b. Point-nerf: Point-based neural radiance fields. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1207] Xu, R., Dong, W., Sharma, A., and Kaess, M. 2022c (Oct.). Learned Depth Estimation of 3D Imaging Radar for Indoor Mapping. In: *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1208] Xu, Wei, Cai, Yixi, He, Dongjiao, Lin, Jiarong, and Zhang, Fu. 2022d. Fastlio2: Fast direct lidar-inertial odometry. *IEEE Trans. Robotics*, **38**(4), 2053–2073.
- [1209] Xu, Xuecheng, Lu, Sha, Wu, Jun, Lu, Haojian, Zhu, Qiuguo, Liao, Yiyi, Xiong, Rong, and Wang, Yue. 2023. Ring++: Roto-translation-invariant gram for global localization on a sparse scan map. *IEEE Trans. Robotics*.
- [1210] Yamazaki, Kashu, Hanyu, Taisei, Vo, Khoa, Pham, Thang, Tran, Minh, Doretto, Gianfranco, Nguyen, Anh, and Le, Ngan. 2024. Open-fusion: Real-time open-vocabulary 3d mapping and queryable scene representation. Pages 9411–9417 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1211] Yan, Chi, Qu, Delin, Xu, Dan, Zhao, Bin, Wang, Zhigang, Wang, Dong, and Li, Xuelong. 2024. Gs-slam: Dense visual slam with 3d gaussian splatting. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1212] Yan, Hang, Shan, Qi, and Furukawa, Yasutaka. 2018. RIDI: Robust IMU double integration. Pages 621–636 of: *European Conf. on Computer Vision (ECCV)*.
- [1213] Yang, Fan, Wang, Chen, Cadenas, Cesar, and Hutter, Marco. 2023a. iPlanner: Imperative Path Planning. In: *Robotics: Science and Systems (RSS)*.
- [1214] Yang, H., and Carlone, L. 2019a. A Polynomial-time Solution for Robust Registration with Extreme Outlier Rates. In: *Robotics: Science and Systems (RSS)*.
- [1215] Yang, H., and Carlone, L. 2019b. A Quaternion-based Certifiably Optimal Solution to the Wahba Problem with Outliers. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1216] Yang, H., and Carlone, L. 2020. One Ring to Rule Them All: Certifiably Robust Geometric Perception with Outliers. Pages 18846–18859 of: *Advances in Neural Information Processing Systems (NIPS)*, vol. 33. .
- [1217] Yang, H., Liang, L., Carlone, L., and Toh, K. 2022a. An Inexact Projected Gradient Method with Rounding and Lifting by Nonlinear Programming for Solving Rank-One Semidefinite Relaxation of Polynomial Optimization. *Mathematical Programming (MAPR)*. .
- [1218] Yang, Heng, and Carlone, Luca. 2022. Certifiably optimal outlier-robust geometric perception: Semidefinite relaxations and scalable global optimization. *IEEE Trans. Pattern Anal. Machine Intell.*, **45**(3), 2816–2834.
- [1219] Yang, Heng, Antonante, Pasquale, Tzoumas, Vasileios, and Carlone, Luca. 2020a. Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection. *IEEE Robotics and Automation Letters*, **5**(2), 1127–1134.
- [1220] Yang, Heng, Shi, Jingnan, and Carlone, Luca. 2020b. Teaser: Fast and certifiable point cloud registration. *IEEE Trans. Robotics*, **37**(2), 314–333.
- [1221] Yang, J., Li, H., Campbell, D., and Jia, Y. 2016. Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration. *IEEE Trans. Pattern Anal. Machine Intell.*, **38**(11), 2241–2254.

- [1222] Yang, Jianing, Sax, Alexander, Liang, Kevin J, Henaff, Mikael, Tang, Hao, Cao, Ang, Chai, Joyce, Meier, Franziska, and Feiszli, Matt. 2025. Fast3r: Towards 3d reconstruction of 1000+ images in one forward pass. Pages 21924–21935 of: *Proceedings of the Computer Vision and Pattern Recognition Conference*.
- [1223] Yang, Jiaolong, Li, Hongdong, and Jia, Yunde. 2014. Optimal essential matrix estimation via inlier-set maximization. Pages 111–126 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [1224] Yang, N., Wang, R., Gao, X., and Cremers, D. 2018a. Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect. In *IEEE Robotics and Automation Letters (RA-L) and Int. Conference on Intelligent Robots and Systems (IROS)*, **3**(Oct), 2878–2885.
- [1225] Yang, Nan, Wang, Rui, Stuckler, Jorg, and Cremers, Daniel. 2018b. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry. Pages 817–833 of: *European Conf. on Computer Vision (ECCV)*.
- [1226] Yang, Nan, Stumberg, Lukas von, Wang, Rui, and Cremers, Daniel. 2020c. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. Pages 1281–1292 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1227] Yang, Shuo, Zhang, Zixin, Bokser, Benjamin, and Manchester, Zachary. 2023b. Multi-IMU Proprioceptive Odometry for Legged Robots. Pages 774–779 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1228] Yang, Wen, Gong, Zheng, Huang, Baifu, and Hong, Xiaoping. 2022b. Lidar With Velocity: Correcting Moving Objects Point Cloud Distortion From Oscillating Scanning Lidars by Fusion With Camera. *IEEE Robotics and Automation Letters*, **7**(3).
- [1229] Yang, Xingrui, Li, Hai, Zhai, Hongjia, Ming, Yuhang, Liu, Yuqian, and Zhang, Guofeng. 2022c. Vox-fusion: Dense tracking and mapping with voxel-based neural implicit representation. Pages 499–507 of: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. IEEE.
- [1230] Yang, Yulin. 2024. *Aided Inertial Navigation System: Analysis and Algorithms*. phdthesis, Unviersity of Delaware.
- [1231] Yang, Yulin, and Huang, Guoquan. 2019. Observability Analysis of Aided INS with Heterogeneous Features of Points, Lines and Planes. *IEEE Trans. Robotics*, **35**(6), 399–1418.
- [1232] Yang, Yulin, Geneva, Patrick, Eckenhoff, Kevin, and Huang, Guoquan. 2019a. Degenerate Motion Analysis for Aided INS with Online Spatial and Temporal Calibration. *IEEE Robotics and Automation Letters*, **4**(2), 2070–2077.
- [1233] Yang, Yulin, Geneva, Patrick, Zuo, Xingxing, Eckenhoff, Kevin, Liu, Yong, and Huang, Guoquan. 2019b (May). Tightly-Coupled Aided Inertial Navigation with Point and Plane Features. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1234] Yang, Yulin, Geneva, Patrick, Zuo, Xingxing, and Huang, Guoquan. 2023c. Online Self-Calibration for Visual-Inertial Navigation Systems: Models, Analysis and Degeneracy. *IEEE Trans. Robotics*, May.
- [1235] Yang, Zheyu, Wang, Taoyi, Lin, Yihan, Chen, Yuguo, Zeng, Hui, Pei, Jing, Wang, Jiazheng, Liu, Xue, Zhou, Yichun, Zhang, Jianqiang, Wang, Xin, Lv,

- Xinhao, Zhao, Rong, and Shi, Luping. 2024. A vision chip with complementary pathways for open-world sensing. *Nature*, **629**(8014), 1027–1033.
- [1236] Yannakakis, M. 1981. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, **2**.
- [1237] Yao, Shanliang, Guan, Runwei, Peng, Zitian, Xu, Chenhang, Shi, Yilu, Ding, Weiping, Lim, Eng Gee, Yue, Yong, Seo, Hyungjoon, Man, Ka Lok, Ma, Jieming, Zhu, Xiaohui, and Yue, Yutao. 2025. Exploring Radar Data Representations in Autonomous Driving: A Comprehensive Review. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, 1–25.
- [1238] Yariv, Lior, Gu, Jiatao, Kasten, Yoni, and Lipman, Yaron. 2021. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems (NIPS)*.
- [1239] Ye, Botao, Liu, Sifei, Xu, Haofei, Xuetong, Li, Pollefeys, Marc, Yang, Ming-Hsuan, and Songyou, Peng. 2025. No Pose, No Problem: Surprisingly Simple 3D Gaussian Splats from Sparse Unposed Images. *Intl. Conf. on Learning Representations (ICLR)*.
- [1240] Ye, Chengxi, Mitrokhin, Anton, Parameshwara, Chethan, Fermüller, Cornelia, Yorke, James A., and Aloimonos, Yiannis. 2020. Unsupervised Learning of Dense Optical Flow, Depth and Egomotion with Event-Based Sensors. Pages 5831–5838 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1241] Yi, Brent, Lee, Michelle A, Kloss, Alina, Martín-Martín, Roberto, and Bohg, Jeannette. 2021. Differentiable factor graph optimization for learning smoothers. Pages 1339–1345 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1242] Yi, Kwang Moo, Verdie, Yannick, Lepetit, Vincent, and Fua, Pascal. 2016. LIFT: Learned Invariant Feature Transform. In: *European Conf. on Computer Vision (ECCV)*.
- [1243] Yilmaz, Alper, Javed, Omar, and Shah, Mubarak. 2006. Object tracking: A survey. *Acm computing surveys (CSUR)*, **38**(4), 13–es.
- [1244] Yin, Huan, Xu, Xuecheng, Wang, Yue, and Xiong, Rong. 2021a. Radar-to-Lidar: Heterogeneous Place Recognition via Joint Learning. *Frontiers in Robotics and AI*, **8**.
- [1245] Yin, Huan, Chen, Runjian, Wang, Yue, and Xiong, Rong. 2021b. Rall: end-to-end radar localization on lidar map using differentiable measurement model. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, **23**(7), 6737–6750.
- [1246] Yin, Huan, Xu, Xuecheng, Lu, Sha, Chen, Xieyuanli, Xiong, Rong, Shen, Shaojie, Stachniss, Cyrill, and Wang, Yue. 2024. A Survey on Global LiDAR Localization: Challenges, Advances and Open Problems. *Intl. J. of Computer Vision*, 1–33.
- [1247] Yin, Jie, Li, Ang, Li, Tao, Yu, Wenxian, and Zou, Danping. 2022. M2DGR: A Multi-Sensor and Multi-Scenario SLAM Dataset for Ground Robots. *IEEE Robotics and Automation Letters*, **7**(2), 2266–2273.
- [1248] Yin, Peng, Zhao, Shiqi, Lai, Haowen, Ge, Ruohai, Zhang, Ji, Choset, Howie, and Scherer, Sebastian. 2023. Automerge: A framework for map assembling and smoothing in city-scale environments. *IEEE Trans. Robotics*.
- [1249] Yokoyama, N., Ha, S., Batra, D., Wang, J., and Bucher, B. 2024. VLFM:

- Vision-Language Frontier Maps for Zero-Shot Semantic Navigation. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1250] Yokozuka, M., Koide, K., Oishi, S., and Banno, A. 2021. LiTAMIN2: Ultra Light LiDAR-Based SLAM Using Geometric Approximation Applied with KL-Divergence. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1251] Yoon, David, Tang, Tim, and Barfoot, Timothy. 2019. Mapless online detection of dynamic objects in 3d lidar. Pages 113–120 of: *Conf. on Computer and Robot Vision (CRV)*. IEEE.
- [1252] Youm, Donghoon, Oh, Hyunsik, Choi, Suyoung, Kim, Hyeongjun, and Hwangbo, Jemin. 2024. Legged Robot State Estimation With Invariant Extended Kalman Filter Using Neural Measurement Network. *arXiv preprint*.
- [1253] Yu, X., and Yang, H. 2024. SIM-Sync: From Certifiably Optimal Synchronization Over the 3D Similarity Group to Scene Reconstruction With Learned Depth. *IEEE Robotics and Automation Letters*, **9**(5), 4471–4478.
- [1254] Yu, Z., Feng, C., Liu, M., and Ramalingam, S. 2017. CASENet: Deep Category-Aware Semantic Edge Detection. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1255] Yuan, Chongjian, Lin, Jiarong, Zou, Zuhao, Hong, Xiaoping, and Zhang, Fu. 2023. STD: Stable triangle descriptor for 3d place recognition. Pages 1897–1903 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [1256] Yuan, Chongjian, Lin, Jiarong, Liu, Zheng, Wei, Hairuo, Hong, Xiaoping, and Zhang, Fu. 2024. BTC: A Binary and Triangle Combined Descriptor for 3D Place Recognition. *IEEE Trans. Robotics*, **40**, 1580–1599.
- [1257] Yuan, Wenzhen, and Ramalingam, Srikuamar. 2016. Fast Localization and Tracking using Event Sensors. Pages 4564–4571 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1258] Yugay, Vladimir, Li, Yue, Gevers, Theo, and Oswald, Martin R. 2023. Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting. *arXiv preprint*.
- [1259] Yun, Seungsang, Jung, Minwoo, Kim, Jeongyun, Jung, Sangwoo, Cho, Younghun, Jeon, Myung-Hwan, Kim, Giseop, and Kim, Ayoung. 2022. STheReO: Stereo Thermal Dataset for Research in Odometry and Mapping. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1260] Zaganidis, A., Sun, L., Duckett, T., and Cielniak, G. 2018. Integrating deep semantic segmentation into 3-d point cloud registration. *IEEE Robotics and Automation Letters*, **3**(4), 2942–2949.
- [1261] Zang, Yuhang, Li, Wei, Zhou, Kaiyang, Huang, Chen, and Loy, Chen Change. 2022. Open-Vocabulary DETR with Conditional Matching. Pages 106–122 of: Avidan, Shai, Brostow, Gabriel, Cissé, Moustapha, Farinella, Giovanni Maria, and Hassner, Tal (eds), *European Conf. on Computer Vision (ECCV)*.
- [1262] Ze, Yanjie, Yan, Ge, Wu, Yueh-Hua, Macaluso, Annabella, Ge, Yuying, Ye, Jianglong, Hansen, Nicklas, Li, Li Erran, and Wang, Xiaolong. 2023. Multi-Task Real Robot Learning with Generalizable Neural Feature Fields. In: *Conf. on Robot Learning (CoRL)*.
- [1263] Zender, Hendrik, Mozos, Óscar Martínez, Jensfelt, Patric, Kruijff, Geert Jan M, and Burgard, Wolfram. 2008. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, **56**(6), 493–502.

- [1264] Zhan, Zitong, Gao, Dasong, Lin, Yun-Jou, Xia, Youjie, and Wang, Chen. 2024. iMatching: Imperative Correspondence Learning. In: *European Conf. on Computer Vision (ECCV)*.
- [1265] Zhang, C., Delitzas, A., Wang, F., Zhang, R., Ji, X., Pollefeys, M., and Engelmann, F. 2025a. Open-Vocabulary Functional 3D Scene Graphs for Real-World Indoor Spaces. In: *arXiv preprint*.
- [1266] Zhang, J., and Singh, S. 2014. LOAM: Lidar Odometry and Mapping in Real-time. In: *Robotics: Science and Systems (RSS)*.
- [1267] Zhang, Jun, Henein, Mina, Mahony, Robert, and Ila, Viorela. 2020. VDO-SLAM: A visual dynamic object-aware SLAM system. *arXiv preprint*.
- [1268] Zhang, Jun, Zhuge, Huayang, Wu, Zhenyu, Peng, Guohao, Wen, Mingxing, Liu, Yiyao, and Wang, Danwei. 2023a. 4DRadarSLAM: A 4D Imaging Radar SLAM System for Large-scale Environments based on Pose Graph Optimization. Pages 8333–8340 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1269] Zhang, Jun, Xiao, Renxiang, Li, Heshan, Liu, Yiyao, Suo, Xudong, Hong, Chaoyu, Lin, Zhongxu, and Wang, Danwei. 2023b. 4DRT-SLAM: Robust SLAM in Smoke Environments using 4D Radar and Thermal Camera based on Dense Deep Learnt Features. Pages 19–24 of: *2023 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*.
- [1270] Zhang, Jun, Zhuge, Huayang, Liu, Yiyao, Peng, Guohao, Wu, Zhenyu, Zhang, Haoyuan, Lyu, Qiyang, Li, Heshan, Zhao, Chunyang, Kircali, Dogan, et al. 2023c. Ntu4dradlm: 4d radar-centric multi-modal dataset for localization and mapping. Pages 4291–4296 of: *IEEE Intl. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE.
- [1271] Zhang, Lintong, Digumarti, Tejaswi, Tinchev, Georgi, and Fallon, Maurice. 2023d. InstaLoc: One-shot Global Lidar Localisation in Indoor Environments through Instance Learning. In: *Robotics: Science and Systems (RSS)*.
- [1272] Zhang, Mike, Qu, Kaixian, Patil, Vaishakh, Cadena, Cesar, and Hutter, Marco. 2025b. Tag Map: A Text-Based Map for Spatial Reasoning and Navigation with Large Language Models. Pages 2120–2146 of: Agrawal, Pukit, Kroemer, Oliver, and Burgard, Wolfram (eds), *Conf. on Robot Learning (CoRL)*. Proceedings of Machine Learning Research, vol. 270. PMLR.
- [1273] Zhang, Ming, Zhang, Mingming, Chen, Yiming, and Li, Mingyang. 2021a. IMU Data Processing For Inertial Aided Navigation: A Recurrent Neural Network Based Approach. Pages 3992–3998 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1274] Zhang, Tingrui, Wang, Jingping, Xu, Chao, Gao, Alan, and Gao, Fei. 2023e. Continuous Implicit SDF Based Any-Shape Robot Trajectory Optimization. Pages 282–289 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1275] Zhang, Xiaoshuai, Kundu, Abhijit, Funkhouser, Thomas, Guibas, Leonidas, Su, Hao, and Genova, Kyle. 2023f. Nerflets: Local Radiance Fields for Efficient Structure-Aware 3D Scene Representation from 2D Supervision. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1276] Zhang, Z., Suleiman, A., Carlone, L., Sze, V., and Karaman, S. 2017. Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Ap-

- proach. In: *Robotics: Science and Systems (RSS)*. , highlighted in the MIT News: .
- [1277] Zhang, Zelin, Yezzi, Anthony, and Gallego, Guillermo. 2023g. Formulating Event-based Image Reconstruction as a Linear Inverse Problem with Deep Regularization using Optical Flow. *IEEE Trans. Pattern Anal. Machine Intell.*, **45**(7), 8372–8389.
- [1278] Zhang, Zhengyou. 1994. Iterative point matching for registration of free-form curves and surfaces. *Intl. J. of Computer Vision*, **13**, 119–152.
- [1279] Zhang, Zhongyang, Cui, Shuyang, Chai, Kaidong, Yu, Haowen, Dasgupta, Subhasis, Mahbub, Upal, and Rahman, Tauhidur. 2024. V2CE: Video to Continuous Events Simulator. Pages 12455–12461 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1280] Zhang, Zichao, and Scaramuzza, Davide. 2018. A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. Pages 7244–7251 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1281] Zhang, Zichao, Sattler, Torsten, and Scaramuzza, Davide. 2021b. Reference Pose Generation for Long-term Visual Localization via Learned Features and View Synthesis. *Intl. J. of Computer Vision*, **129**(4), 821–844.
- [1282] Zhao, Ji, Xu, Wanting, and Kneip, Laurent. 2020a. A Certifiably Globally Optimal Solution to Generalized Essential Matrix Estimation. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1283] Zhao, Shibo, Wang, Peng, Zhang, Hengrui, Fang, Zheng, and Scherer, Sebastian. 2020b. Tp-tio: A robust thermal-inertial odometry with deep thermalpoint. Pages 4505–4512 of: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [1284] Zhao, Shibo, Zhang, Hengrui, Wang, Peng, Nogueira, Lucas, and Scherer, Sebastian. 2021. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. Pages 8729–8736 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE.
- [1285] Zhao, Shibo, Gao, Yuanjun, Wu, Tianhao, Singh, Damanpreet, Jiang, Rushan, Sun, Haoxiang, Sarawata, Mansi, Qiu, Yuheng, Whittaker, Warren, Higgins, Ian, Du, Yi, Su, Shaoshu, Xu, Can, Keller, John, Karhade, Jay, Nogueira, Lucas, Saha, Sourojit, Zhang, Ji, Wang, Wenshan, Wang, Chen, and Scherer, Sebastian. 2023. *SubT-MRS Dataset: Pushing SLAM Towards All-weather Environments*.
- [1286] Zheng, Jianhao, Zhu, Zihan, Bieri, Valentin, Pollefeys, Marc, Peng, Songyou, and Iro, Armeni. 2025. WildGS-SLAM: Monocular Gaussian Splatting SLAM in Dynamic Environments. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1287] Zheng, Y., Sugimoto, S., and Okutomi, M. 2011. Deterministically maximizing feasible subsystem for robust model fitting with unit norm constraint. Pages 1825–1832 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1288] Zhi, Shuaifeng, Laidlow, Tristan, Leutenegger, Stefan, and Davison, Andrew J. 2021. In-place scene labelling and understanding with implicit scene representation. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [1289] Zhi, Shuaifeng, Sucar, Edgar, Mouton, Andre, Haughton, Iain, Laidlow, Tristan, and Davison, Andrew J. 2022. ilabel: Revealing objects in neural fields. *IEEE Robotics and Automation Letters*, **8**(2), 832–839.

- [1290] Zhong, Xingguang, Pan, Yue, Behley, Jens, and Stachniss, Cyrill. 2023. Shine-mapping: Large-scale 3d mapping using sparse hierarchical implicit neural representations. Pages 8371–8377 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1291] Zhou, Haoyin, and Jayender, Jagadeesan. 2021. EMDQ-SLAM: Real-time high-resolution reconstruction of soft tissue surface from stereo laparoscopy videos. Pages 331–340 of: *Intl. Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*.
- [1292] Zhou, Kun, Hou, Qiming, Wang, Rui, and Guo, Baining. 2008. Real-time kd-tree construction on graphics hardware. *ACM Trans. on Graphics (TOG)*, **27**(5), 1–11.
- [1293] Zhou, Qian-Yi, and Koltun, Vladlen. 2013. Dense scene reconstruction with points of interest. *ACM Trans. on Graphics (TOG)*, **32**(4).
- [1294] Zhou, Q.Y., Park, J., and Koltun, V. 2016. Fast global registration. Pages 766–782 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [1295] Zhou, Shenghao, Katragadda, Saimouli, and Huang, Guoquan. 2025 (May). Learning IMU Bias with Diffusion Model. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [1296] Zhou, Tinghui, Brown, Matthew, Snavely, Noah, and Lowe, David G. 2017. Unsupervised learning of depth and ego-motion from video. Pages 1851–1858 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1297] Zhou, Xingyi, Karpur, Arjun, Luo, Linjie, and Huang, Qixing. 2018a. StarMap for Category-Agnostic Keypoint and Viewpoint Estimation. In: *European Conf. on Computer Vision (ECCV)*.
- [1298] Zhou, Yi, Gallego, Guillermo, Rebecq, Henri, Kneip, Laurent, Li, Hong-dong, and Scaramuzza, Davide. 2018b. Semi-Dense 3D Reconstruction with a Stereo Event Camera. Pages 242–258 of: *European Conf. on Computer Vision (ECCV)*.
- [1299] Zhou, Yi, Gallego, Guillermo, and Shen, Shaojie. 2021. Event-based Stereo Visual Odometry. *IEEE Trans. Robotics*, **37**(5), 1433–1450.
- [1300] Zhu, Alex Zihao, Atanasov, Nikolay, and Daniilidis, Kostas. 2017. Event-based Visual Inertial Odometry. Pages 5816–5824 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1301] Zhu, Alex Zihao, Thakur, Dinesh, Ozaslan, Tolga, Pfrommer, Bernd, Kumar, Vijay, and Daniilidis, Kostas. 2018. The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception. *IEEE Robotics and Automation Letters*, **3**(3), 2032–2039.
- [1302] Zhu, Alex Zihao, Yuan, Liangzhe, Chaney, Kenneth, and Daniilidis, Kostas. 2019. Unsupervised Event-based Learning of Optical Flow, Depth, and Ego-motion. Pages 989–997 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1303] Zhu, Alex Zihao, Wang, Ziyun, Khant, Kaung, and Daniilidis, Kostas. 2021. EventGAN: Leveraging Large Scale Image Datasets for Event Cameras. Pages 1–11 of: *IEEE Int. Conf. Comput. Photography (ICCP)*.
- [1304] Zhu, Liyuan, Huang, Shengyu, Schindler, Konrad, and Armeni, Iro. 2024. Living scenes: Multi-object relocalization and reconstruction in changing 3d environments. Pages 28014–28024 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [1305] Zhu, Liyuan, Li, Yue, Sandström, Erik, Huang, Shengyu, Schindler, Konrad, and Armeni, Iro. 2025. LoopSplat: Loop Closure by Registering 3D Gaussian Splats. In: *Intl. Conf. on 3D Vision (3DV)*.
- [1306] Zhu, Zihan, Peng, Songyou, Larsson, Viktor, Xu, Weiwei, Bao, Hujun, Cui, Zhaopeng, Oswald, Martin R., and Pollefeys, Marc. 2022. NICE-SLAM: Neural Implicit Scalable Encoding for SLAM. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [1307] Zhuang, Yuan, Wang, Binliang, Huai, Jianzhu, and Li, Miao. 2023a. 4D iRIOM: 4D Imaging Radar Inertial Odometry and Mapping. *IEEE Robotics and Automation Letters*, **8**(6), 3246–3253.
- [1308] Zhuang, Ziwen, Fu, Zipeng, Wang, Jianren, Atkeson, Christopher G., Schwertfeger, Sören, Finn, Chelsea, and Zhao, Hang. 2023b. Robot Parkour Learning. Pages 73–92 of: Tan, Jie, Toussaint, Marc, and Darvish, Kourosh (eds), *Conf. on Robot Learning (CoRL)*. Proceedings of Machine Learning Research, vol. 229. PMLR.
- [1309] Zou, Xueyan, Yang, Jianwei, Zhang, Hao, Li, Feng, Li, Linjie, Wang, Jianfeng, Wang, Lijuan, Gao, Jianfeng, and Lee, Yong Jae. 2023. Segment everything everywhere all at once. In: *Conf. Neural Information Processing Systems (NIPS)*.
- [1310] Zou, Xueyan, Song, Yuchen, Qiu, Ri-Zhao, Peng, Xuanbin, Ye, Jianglong, Liu, Sifei, and Wang, Xiaolong. 2025. 3D-Spatial MultiModel Memory. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [1311] Zuckerman, David. 2006 (May). Linear degree extractors and the inapproximability of max clique and chromatic number. Pages 681–690 of: *ACM Symp. on Theory of Computing (STOC)*.
- [1312] Zuo, X., Xie, J., Liu, Y., and Huang, Guoquan. 2017 (Sept.). Robust Visual SLAM with Point and Line Features. Pages 1775–1782 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [1313] Zuo, Xingxing, Samangouei, Pouya, Zhou, Yunwen, Di, Yan, and Li, Mingyang. 2024a. Fmgs: Foundation model embedded 3d gaussian splatting for holistic 3d scene understanding. *Intl. J. of Computer Vision*.
- [1314] Zuo, Yi-Fan, Xu, Wanting, Wang, Xia, Wang, Yifu, and Kneip, Laurent. 2024b. Cross-Modal Semidense 6-DOF Tracking of an Event Camera in Challenging Conditions. *IEEE Trans. Robotics*, **40**, 1600–1616.
- [1315] Zwicker, Matthias, Pfister, Hanspeter, van Baar, Jeroen, and Gross, Markus. 2001. Surface Splatting. In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- [1316] Zwicker, Matthias, Pfister, Hanspeter, Van Baar, Jeroen, and Gross, Markus. 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, **8**(3), 223–238.
- [1317] Źywanowski, Kamil, Banaszczyk, Adam, Nowicki, Michał R., and Komorowski, Jacek. 2022. MinkLoc3D-SI: 3D LiDAR Place Recognition With Sparse Convolutions, Spherical Coordinates, and Intensity. *IEEE Robotics and Automation Letters*, **7**(2), 1079–1086.



## Author index

- Abhinav Valada, 489  
Andrew J. Davison, 396, 520  
Arash Asgharivaskasi, 453  
Ayoung Kim, 5, 120, 185, 223, 249, 361  
Boris Chidlovskii, 366  
Cédric Le Gentil, 303  
Cesar Cadena, 489  
Chen Wang, 102  
Christoffer Heckman, 249  
Connor Holmes, 150  
Cyrill Stachniss, 120  
Daniel Cremers, 5, 185, 193, 361, 366, 416  
David M. Rosen, 150  
Davide Scaramuzza, 281, 303  
Dominic Maggio, 489  
Felix Wimbauer, 366  
Frank Dellaert, 5, 21, 54, 185, 193, 303, 361  
Fu Zhang, 223  
Gamini Dissanayake, 150  
Giseop Kim, 223  
Guillermo Gallego, 281  
Guoquan (Paul) Huang, 303  
Helen Oleynikova, 120  
Heng Yang, 76  
Henrik Andreasson, 249  
Henrik I. Christensen, 453  
Hidenobu Matsuki, 396  
Jakob Engel, 193  
Javier Civera, 193, 416  
Javier Hidalgo-Carrió, 281  
Jens Behley, 120, 223, 453  
Ji Zhang, 223  
Jia Deng, 366  
Jingnan Shi, 76  
John Leonard, 453  
Jose Luis Blanco-Claraco, 54  
Jose Maria Martinez Montiel, 416  
Jose Neira, 416  
Josh Mangelson, 76  
Juan D. Tardós, 193  
Jérôme Revaud, 366  
Kasra Khosoussi, 150  
Kevin Doherty, 453  
Krishna Murthy Jatavallabhula, 102  
Liam Paull, 489  
Lionel Ott, 120  
Luca Carlone, 5, 76, 150, 185, 303, 361, 453, 489  
Lukas Schmid, 416  
Marc Pollefeys, 361  
Marco Camurri, 332  
Margarita Chli, 193  
Martin Büchner, 489  
Martin Magnusson, 249  
Matfas Mattamala, 332  
Maurice Fallon, 223  
Michael Kaess, 21, 54, 249  
Mustafa Mukadam, 102  
Nathan Hughes, 453  
Niko Sunderhauf, 76  
Nikolay Atanasov, 453  
Paul Newman, 249  
Sacha Morin, 489  
Shibo Zhao, 223  
Shoudong Huang, 416  
Stefan Leutenegger, 193  
Teresa Vidal-Calleja, 120, 303  
Timothy Barfoot, 5, 21, 54, 150, 185, 249, 361  
Victor Reijgwart, 120  
Yun Chang, 76, 453  
Zachary Teed, 366

## Subject index

- CRLB, 175  
FIM, 175  
IMU preintegration, 308  
PCM, 83  
linear least squares, 160  
2D, 121, 127, 225, 238, 437, 456–460, 472  
2D point features, 7  
2D-2D correspondences, 80  
3D, 121, 123, 225, 226, 238, 455–464, 467, 473–475, 486  
3D Gaussian Splatting (3DGS), 409, 452  
3D pointmaps, 387  
3D scene graph, 486  
3D structure, 437  
3D-3D correspondences, 83  
4D, 421, 445  
a posteriori, 101, 166, 468  
a priori, 131  
absolute optical encoders, 340  
Absolute Trajectory Error (ATE), 301  
accelerometer, 304  
adaptive rendering, 295  
Aided Inertial Navigation Systems (AINS), 305  
algebraic connectivity, 159  
Amplitude Modulated Continuous Wave (AMCW), 225  
Analog-to-Digital Converter (ADC), 280  
anchor, 22, 222, 245, 279, 350  
AR/VR, 362  
articulated objects, 428, 431  
asynchronous, 64, 189, 283, 286, 289, 302, 317, 541, 542  
Asynchronous Time-based Image Sensor (ATIS), 284  
awareness, 238  
axis, 157  
batch optimization, 205  
Bayes tree, 40, 47, 48  
bearing, 170  
Bidirectional Encoder Representations from Transformers (BERT), 492, 496  
bilevel optimization, 104  
Black-Rangarajan (B-R) duality, 91  
body manipulator Jacobian, 349  
bounding box, 135, 455, 459–461, 499, 502, 504, 505, 514  
Bounding Volume Hierarchies (BVH), 134  
branch-and-bound, 99  
Brown-Conrady, 199  
bundle adjustment (BA), 8–10, 25, 26, 87, 103, 105, 119, 194–196, 210, 211, 214, 216, 223, 293, 328, 366, 380, 381, 433, 436, 438, 450, 451, 541  
Burér-Monteiro factorization, 163  
calibration, 122, 180, 190, 199, 204, 249, 280, 289, 306, 329, 386, 394  
camera pose prediction, 370  
center of pressure, 346  
certifiable algorithms, 100  
certifiable optimal solvers, 30  
certifiably correct, 152, 166  
certifiably globally optimal, 151  
certifiably optimal, 152  
certifiably optimal algorithms, 150  
certificate matrix, 168  
change detection, 248, 435, 437, 442  
chirp, 253  
Cholesky factor, 32  
Cholesky factorization, 32  
chordal, 48  
clique tree, 48  
Column Approximate Minimum Degree permutation (COLAMD), 40  
compass, 304  
complementary, 335  
completion, 302  
complexity, 288  
computation, 150  
computational, 164  
computational certificate, 166  
Computer Aided Design (CAD), 457

- concentration, 157  
 conditionally independent, 469  
 conditions for exactness, 181  
 connection Laplacian, 160  
 connectivity, 177  
 consensus maximization, 81  
 consensus set, 82  
 Conservative Filtering for Efficient and Accurate (CFEAR), 267  
 consistency function, 83, 84  
 Constant False Alarm Rate (CFAR), 260  
 contact estimation, 334  
 context features, 378  
 continuous, 421, 466, 468  
 continuous-time, 54  
 continuous-time  
 trajectory, 63  
 contrast maximization, 292  
 contrastive, 374  
 Contrastive Language-Image Pre-training (CLIP), 494, 495, 497  
 contrastive learning, 492  
 control nodes, 479  
 convex, 154  
 convex relaxation, 152, 154  
 Convolutional Neural Network (CNN), 265, 270, 491, 492  
 correlation features, 378  
 correspondence search, 230  
 Cramér-Rao lower bound, 175  
 cubes, 132  
 D3VO, 372  
 data association, 9, 15, 17, 19, 21, 29, 77, 137, 171, 196, 230, 266, 278, 285, 287–289, 291, 292, 417, 423, 432, 443, 448, 452, 454, 466, 468, 469, 471, 525  
 dead reckoning, 343  
 decision variable, 163  
 Degree of Freedom (DoF), 280  
 Deeper Into Neural Networks (DINO), 494–496  
 deformable, 417, 420  
 Degree of Freedom (DoF), 335, 336, 338, 352, 427, 441  
 dense open-vocabulary maps, 500  
 dense reconstruction, 265, 428, 437  
 dense surface, 437  
 depth, 370–373  
 depth prediction, 369  
 differentiable bundle adjustment, 380, 381  
 Differentiable Rendering (DR), 399, 400  
 differentiation on manifold, 110  
 direct methods, 7, 207, 287, 430  
 direct tracking, 219  
 direct vs. indirect, 287  
 directed graph, 159  
 discrete, 466, 468  
 discrete-continuous, 468  
 distance, 123, 139  
 distortion, 198  
 distributed computation, 540  
 Doppler effect, 426  
 double sphere, 199  
 DPVO, 384  
 DROID-SLAM, 380  
 dull, dangerous, and dirty, 356  
 DuSt3R, 386  
 DVS, 283  
 DVSO, 372  
 dynamic, 417, 421, 428  
 Dynamic and Active-Pixel Vision Sensor (DAVIS), 284  
 dynamic and deformable SLAM, 418  
 dynamic covariance scaling, 94  
 dynamic deformable graph, 450  
 dynamic maps, 445  
 EKF-SLAM, 15, 349, 448  
 embedded deformation graph, 479  
 embody, 356  
 epipolar constraint, 80  
 equivariant, 441  
 Euclidean Signed Distance Function (ESDF), 126, 127, 139  
 evaluation, 191  
 event alignment, 292  
 event batches, 287  
 event cameras, 185, 186, 189, 282, 383, 426, 532, 535, 540  
 event generation model, 287  
 event-by-event, 287  
 event-by-event basis, 287  
 events, 283  
 evidence of absence, 435  
 exactness, 168  
 Expectation Minimization (EM), 443  
 explicit surface model, 457  
 exploration-exploitation, 294  
 exploratory trajectory, 447, 448  
 exteroceptive, 7  
 face, 130  
 factor, 23, 24, 27, 308  
 factor graph, 17–27, 37, 40, 45, 50, 54, 56, 64, 69, 76, 118, 151, 156, 187, 206, 210, 220, 241, 263, 268, 304, 308, 310, 313, 316, 327, 330, 335, 349, 353, 366, 367, 373, 375, 378–380, 383, 386, 427, 428, 443, 455, 540–542, 545  
 feasible approximate, 164, 165  
 feature-based SLAM, 205  
 featuremetric, 539  
 Field Of View (FOV), 190, 226, 269, 270, 272  
 filtering, 206

- Fisher information matrix, 175  
 fisheye camera, 199  
 flow, 378, 382  
 forgetting, 434  
 foundation model, 491  
 frequency map, 445  
 Frequency Modulated Continuous Wave (FMCW), 225–227, 253, 270  
 g-sensitivity, 306  
 Gauss-Newton (GN), 35, 36, 385  
 Gaussian Bayes net, 45  
 Gaussian Belief Propagation, 541  
 Gaussian Process (GP), 63, 68, 70, 73–75, 128, 131, 142, 143, 232, 316, 318, 319  
 Gaussian Process Implicit Surface (GPIS), 143  
 Gaussian Process Occupancy Map (GPOM), 143, 144  
 Gaussian Process Regression, 69  
 generalization, 292  
 Generative Pre-trained Transformer (GPT), 492, 498, 499  
 geodesic distance, 301  
 geodesy, 15  
 geometric primitives, 289  
 global, 164  
 global consistency, 240  
 global minimizer, 155, 162  
 Global Positioning System (GPS), 5, 6, 185, 187, 191, 197, 221, 224, 299, 305, 325, 329, 363, 364  
 global shutter, 200  
 global solvers, 100  
 globally, 150  
 globally aligned, 391  
 globally consistent, 13  
 globally optimal, 155, 162, 164  
 Graduated Non-Convexity (GNC), 89, 94–96, 98–100  
 Graphics Processing Unit (GPU), 129, 135, 434  
 Ground Reaction Force (GRF), 346, 347  
 guaranteed, 164  
 gyroscope, 232, 263, 304–307, 316, 317, 321, 331  
 hash function, 134  
 hash map, 134  
 healthcare, 364  
 Hessian matrix, 32  
 hidden state, 376, 378  
 hierarchy, 174  
 high-dimensional, 151  
 high-dynamic, 421  
 Hilbert Map (HM), 144  
 homogeneous, 56  
 homogeneous point, 56  
 Householder reflection matrix, 33  
 humanoid, 363  
 hybrid discrete-continuous factor graph, 468  
 hybrid vision sensor, 286  
 hypergraphs, 100  
 hyperparameters, 67  
 iKD-tree, 235  
 image reconstruction, 293  
 implicit differentiation, 108  
 implicit surface, 126  
 implicit surface model, 457  
 incremental encoders, 340  
 Incremental Smoothing and Mapping (iSAM), 21, 48, 51, 53  
 Independently Moving Objects (IMO), 298  
 indirect, 265, 287  
 indirect methods, 7  
 industrial automation, 364  
 Inertial Measurement Unit (IMU), 123, 137, 185, 192, 200, 219, 221, 224, 232, 263, 277, 286, 294, 298, 304, 305, 307, 308, 311, 315, 319, 321, 326, 333, 337, 344, 345, 347, 354, 424, 525  
 Inertial Navigation Systems (INS), 304  
 inertial odometry, 305  
 information matrix, 17, 32, 176  
 infrared (IR), 122, 340  
 instance segmentation, 455  
 inter-layer edges, 487  
 interdisciplinary, 302  
 invariance, 238  
 invariant, 441  
 inverse problem, 7  
 isotropic Langevin distribution, 156  
 Iterative Closest Point (ICP), 136, 218, 227–230, 235, 242  
 Iteratively Reweighted Least Squares (IRLS), 90, 91, 93, 94, 96, 100  
 Jacobian matrix, 337  
 joint estimation, 293  
 joint positions, 336  
 joint velocities, 337  
 junction tree, 48  
 Kannala-Brandt, 199  
 kernel function, 63, 67  
 kernel matrix, 67  
 kernel trick, 67  
 keyframes, 206  
 keypoint descriptor, 200  
 keypoint detector, 200  
 keypoints, 200, 374  
 Laplacian, 158, 159  
 Large Language Model (LLM), 497–499  
 leg odometry, 333, 334, 424  
 Levenberg-Marquardt (LM), 35, 36  
 Lie algebra, 56, 60  
 Lie group adjoint, 62

- Lie group Jacobian, 63
- Lie groups, 56
- lifelong, 421
- lifts, 154
- Light Detection and Ranging (LiDAR), 7, 10, 120–122, 128, 136, 155, 185, 186, 188, 192, 196, 217, 224–228, 230–243, 245–249, 251, 252, 260–262, 266–274, 277–280, 310, 319, 322, 323, 333, 335, 339, 353, 364, 414, 423, 426, 429, 465, 471, 483
- Light Emitting Diode (LED), 340, 342
- LightLight Detection and Ranging (LiDAR), 122
- linear, 154
- linear least squares, 30, 31
- linearized, 30
- list decodable regression, 181
- local, 151, 152, 154, 164, 317, 386
- local BA, 208
- local consistency, 442
- local support, 63, 64
- localization, 5, 428
- location, 156
- long-term, 419–421, 432
- long-term dynamics, 419, 432
- loop closure, 8–10, 25, 77, 86, 132, 185, 186, 195, 196, 208, 209, 219, 222, 237, 240, 242, 243, 261, 274–276, 305, 326, 335, 382, 386, 407, 413, 433, 439, 454, 471, 472, 479, 487, 495, 536
- low-dynamic, 421
- low-rank, 162
- Lower-Diagonal-Upper (LDU), 33, 39
- Lower-Upper (LU), 32
- M-estimation, 87, 150
- M-step, 470
- manifold, 54
- map, 437
- map cleaning, 436
- map representation, 428
- map summarization, 434
- mapping, 5, 428
- Maps of Dynamics (MoD), 445
- marginal distribution, 435
- marginalization, 435
- Markov Random Field (MRF), 38, 39, 45
- Markovian, 69
- mask module, 370, 425
- MASt3R, 392
- MASt3R-SfM, 393
- MASt3R-SLAM, 394
- matrix Lie groups, 56
- Maximum A posteriori (MAP), 17, 21, 26–28, 30, 39, 40, 46, 47, 51, 56, 58, 64, 66, 72, 75
- maximum clique, 84, 85
- maximum consensus, 87
- maximum likelihood, 17
- measurement, 333
- measurement Jacobian, 31
- measurement model, 29
- measurement prediction function, 379
- mechanical LiDAR, 225
- memory, 13
- mesh, 130, 138, 461
- metric vs. topological maps, 13
- metric-semantic point cloud and surfel map representations, 471
- metric-semantic point cloud map, 471
- metric-semantic surfel map, 471
- minimal set, 81
- minimal solvers, 9, 81
- mmWave, 253
- mode, 156
- moment (or Lasserre’s) relaxation, 170, 173
- MonoRec, 369
- motion estimation, 334
- motion model, 29
- motion-capture system, 298
- motion-distorted, 64
- Moving Least Squares (MLS), 138
- Moving-Object Segmentation (MOS), 429
- Moving-Object Tracking (MOT), 428, 430
- moving-object tracking (MOT), 427
- multi-class, 455
- multi-class log odds, 474
- multi-class octree, 476
- multi-disciplinarity of SLAM, 10
- multi-frontal QR factorization, 45
- multi-hypothesis, 440, 470
- multi-instance dynamic (MID) SLAM, 427
- multi-robot mapping, 419
- multi-session, 437, 441
- multi-session mapping, 419, 437
- Multiple Input, Multiple Output (MIMO), 258
- multiplicative isotropic Langevin noise, 157
- multivariate Gaussian density, 28
- MUSt3R, 394
- Natural Language Processing (NLP), 492, 496
- Neural Fields, 404
- Neural Radiance Fields (NeRF), 401
- Neural Radiance Fields (NeRF), 145, 146, 149, 452, 464
- neuromorphic computing, 302
- nodes, 159
- noise model, 296
- non-rigid scene models, 448
- nonconvex, 151, 155
- nonlinear least squares, 30, 76
- nonparametric, 63, 67
- Normal Distributions Transform (NDT), 130, 137, 230
- normal equations, 31, 32, 67

- NP-hard, 151
- NRSfM, 423, 446–448
- object detection, 455
- object landmark, 456
- object re-localization, 439
- observability analysis, 320
- observability matrix, 320, 324
- observation, 419
- occupancy, 123, 124, 139
- octant, 134
- octree, 140
- odometry, 7, 10, 11, 13, 25, 76, 77, 96, 98, 105, 123, 151, 167, 185, 186, 195, 196, 227, 228, 231, 232, 234, 235, 250, 260–262, 265, 266, 268, 275, 278, 298, 304, 319, 326, 330, 333, 372, 384, 386, 416, 424, 428, 443, 472, 526
- online vs. offline, 422, 448
- open set, 440
- open-vocabulary object maps, 504
- open-world representations, 490
- optical absolute encoder, 340
- optical flow, 196, 287, 366, 374, 375, 378, 382, 386, 451, 538
- optical incremental encoder, 340
- orthonormal frame, 163
- pairwise consistency constraints, 84
- pairwise consistency maximization, 84
- panoptic segmentation, 455
- parametric, 63, 430
- parametric model, 430
- Partial Differential Equation (PDE), 138
- partial measurements, 438
- partial observations, 438
- particle depletion, 17
- particle-filter, 16
- patch graph, 384
- perceptual aliasing, 14, 77, 432, 440
- persistent, 421, 445
- persistent representation, 5
- Perspective-n-Point (PnP), 58
- photogrammetry, 194
- photometric calibration, 199
- photometric loss, 210
- physical, 311
- pinhole camera model, 198
- place recognition, 14, 77, 186, 236–238, 250, 268–272, 279, 433, 439
- places, 486
- plus, 315
- point cloud undistortion, 231
- point-to-line distance, 229
- point-to-plane distance, 229
- point-to-point distance, 229
- pointmap, 387
- points, 128, 136
- polynomial optimization problem, 172
- pose, 55, 371–373, 382
- pose graph, 156, 268, 274, 275, 353, 479, 481, 487
- pose-graph optimization (PGO), 10, 25, 26, 105, 137, 150–153, 155–157, 162, 164–166, 179–181, 210, 211, 242, 481, 482, 487, 541
- pose-only BA, 208
- poses, 155, 161
- PoseSLAM, 21
- positive-semidefinite, 153
- Powell's dogleg (PDL), 36
- preintegrated IMU measurements, 311
- preintegrated contact noise, 351
- preintegrated contacts, 351
- preintegrated measurement model, 313
- preintegrated position measurement, 313
- preintegrated rotation measurement, 312
- preintegrated velocity, 351
- preintegrated velocity measurement, 312
- preintegrated velocity noise, 352
- preintegration, 308
- Principal Component Analysis (PCA), 138, 271, 276, 457
- probabilistic graphical models, 23
- probabilistic inference, 23
- probabilities, 125, 470
- probability, 124
- Probability Density Function (PDF), 22, 473
- Probability Mass Function (PMF), 473, 474
- processor architecture, 528
- projection, 198
- projective, 126
- proprioceptive, 7
- proprioceptive and exteroceptive sensors, 7
- QR-factorization, 33
- Quadratically Constrained Quadratic Program (QCQP), 153
- quadratics, 153
- quadric surface, 458
- quartic, 171
- quasi-static, 421
- quinary, 65
- radar, 7, 64, 171, 185, 189, 225, 249, 250, 302, 421, 426
- Radar Cross Section (RCS), 253, 254, 258, 271, 274
- radar datacube, 252
- radargram, 252
- Radial Basis Function (RBF), 138
- radial-tangential, 198
- RANDom SAmple Consensus (RANSAC), 80, 81
- range sensor, 120
- range-category observation, 473
- rank- $d$  approximation, 165
- rank-1, 153

- recall, 139
- Recurrent All Pairs Field Transforms (RAFT), 375
- reduced incidence matrix, 161
- redundancy, 177
- redundant constraints, 173
- regularization, 375
- regularizer term, 66
- Relative Pose Error (RPE), 301
- remission, 226
- representation, 422
- reprojection error, 195, 199, 204, 207, 293, 427
- retracting, 60
- RGB-D, 120–122, 215, 217, 238, 300, 383, 413, 446, 449, 500, 545
- Riemannian optimization, 60
- Riemannian Staircase, 164
- rigid body, 430
- rigid moving body, 427
- rigid moving objects, 430
- rigid scene model, 424, 427, 430
- robot parkour, 356
- robot pose, 422
- robotics foundation model, 516
- robust cost function, 425
- robust kernels, 204
- robust loss, 87
- rolling shutter, 200
- rotation matrix, 55, 161
- scan, 227
- scan registration, 228
- scan-to-map odometry, 231, 235
- scan-to-scan odometry, 231, 235
- scene differencing, 437, 443
- scene prediction, 445
- Schur complement, 168, 213
- SE-Sync, 153, 155, 165
- search and rescue, 364
- segmentation, 298
- segmentation mask, 461
- self-driving, 363
- semantic consistency, 438, 439
- semantic edges, 455
- semantic keypoint, 455, 457
- semantic landmark, 457
- semantic segmentation, 425, 429, 438, 455
- semi-static, 421
- semidefinite program, 101, 154
- semidefinite relaxation, 150
- sensor degradation, 77
- sensor fusion, 302
- Shor's relaxation, 152–154
- short-term, 419–421, 432
- short-term dynamics, 419, 423
- Signal-to-Noise Ratio (SNR), 225
- Signed Distance Function (SDF), 128, 145, 449, 450, 464–467
- Simultaneous Localization and Mapping (SLAM), 5, 8, 11, 16, 21–27, 29, 30, 32, 34, 37, 38, 40, 47, 54–56, 58, 64, 69, 120, 121, 128, 136, 146, 147, 185, 192, 224, 227, 228, 232–236, 240, 246, 248–250, 268, 273, 279, 280, 423, 454–458, 461–463, 466, 478, 490, 494
- Simultaneous Trajectory Estimation And Mapping (STEAM), 25, 26, 69
- SLAM and deep learning, 366
- SLAM and spatial memory, 12
- SLAM and Structure from Motion, 12
- SLAM back-end, 9
- SLAM backend, 243
- SLAM front-end, 9
- SLAM front-end and back-end, 8
- SLAM, localization, mapping, 5
- Smoothing and Mapping (SAM), 21, 33
- SoC, 250–253, 255, 257, 263, 268–270, 272, 273, 276, 277, 279, 280
- SoC radar, 252
- solid-state LiDAR, 226
- sparse, 174, 466
- sparse QR factorization, 45
- sparsity, 286
- Spatial AI, 18, 361, 490
- spatial AI, 521
- spatial perception, 18
- spatio-temporal, 421
- spectral, 176
- spikes, 283
- spinning radar, 252
- splines, 64
- Square-Root Information Filtering (SRIF), 22
- square-root SAM, 34
- stance phase, 334
- state estimation, 335
- static, 417, 428
- Steepest Descent (SD), 34–36
- Stiefel manifold, 163
- Stochastic Differential Equation (SDE), 67–69, 74
- strain gauge, 341
- Structure from Motion (SfM), 194, 195, 409, 423, 446
- Structure from Template (SfT), 447, 448, 450
- structure-exploiting, 162
- sub-symbolic, 483
- submaps, 443
- suboptimality, 154
- surface, 127
- surfels, 129, 137
- swing phase, 334
- symbol grounding, 483
- symmetric rank- $r$  factorization, 163

synchronization, 191  
tangent space, 60  
temporal basis functions, 64  
temporal smoothing, 446  
temporal variation patterns, 445  
tightness, 228  
Time-Of-Flight (TOF), 121, 122, 225  
topological map, 14  
transformation matrix, 55  
transient, 419  
transition function, 68  
translational weight graph, 160  
tree, 134  
triangulated, 48  
truncated, 165  
Truncated Signed Distance Function (TSDF),  
    127, 138–140, 148, 431, 436, 437, 501  
truncation band, 139  
trust-region method, 35  
unbiased, 175  
uncertainty measure, 440  
unconstrained, 163  
undirected graphical model, 38  
unit of coherent change, 437, 438  
unobserved, 437  
unorganized, 122  
unprojection, 199  
unrolled differentiation, 105  
variable elimination, 22  
vertex, 130  
visibility, 436  
Vision Language Model (VLM), 499  
vision transformer, 493  
Visual Effects (VFX), 140  
visual pathway, 284  
visual place recognition, 207, 495  
visual SLAM, 193, 195  
visual-inertial odometry, 220, 286, 305  
visual-inertial SLAM, 221, 305  
volumetric, 437  
volumetric map, 436  
voxels, 131, 139, 436  
wavefront propagation, 132  
world model, 522