

# SLAM Handbook

From Localization and Mapping to Spatial Intelligence

## Edited by

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

**IMPORTANT NOTE ON RELEASE:**

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

## Together with all contributors

Henrik Andreasson	Arash Asgharivaskasi	Nikolay Atanasov
Timothy Barfoot	Jens Behley	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	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	Martin Magnusson
Joshua Mangelson	Hidenobu Matsuki	Matias Mattamala
José M Martínez Montiel	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
Teresa Vidal-Calleja	Chen Wang	Felix Wimbauer
Heng Yang	Fu Zhang	Ji Zhang
Shibo Zhao		



# Contents

*Notation* *page 1*

<b>PART ONE FOUNDATIONS OF SLAM</b>		3
<b>1</b>	<b>Prelude</b>	5
1.1	What is SLAM?	5
1.2	Anatomy of a Modern SLAM System	7
1.3	The Role of SLAM in the Autonomy Architecture	11
1.3.1	Do we really need SLAM for robotics?	12
1.4	Past, Present, and Future of SLAM, and Scope of this Handbook	15
1.4.1	Short History and Scope of this Handbook	15
1.4.2	From SLAM to Spatial AI	18
1.5	Handbook Structure	19
<b>2</b>	<b>Factor Graphs for SLAM</b>	21
2.1	Visualizing SLAM With Factor Graphs	22
2.1.1	A Toy Example	22
2.1.2	A Factor-Graph View	23
2.1.3	Factor Graphs as a Language	25
2.2	From MAP Inference to Least Squares	26
2.2.1	Factor Graphs for MAP Inference	27
2.2.2	Specifying Probability Densities	28
2.2.3	Nonlinear Least Squares	30
2.3	Solving Linear Least Squares	30
2.3.1	Linearization	31
2.3.2	SLAM as Least-Squares	32
2.3.3	Matrix Factorization for Least-Squares	32
2.4	Nonlinear Optimization	34
2.4.1	Steepest Descent	34
2.4.2	Gauss-Newton	35
2.4.3	Levenberg-Marquardt	35

2.4.4	Dogleg Minimization	36
2.5	Factor Graphs and Sparsity	37
2.5.1	The Sparse Jacobian and its Factor Graph	37
2.5.2	The Sparse Information Matrix and its Graph	38
2.5.3	Sparse Factorization	39
2.6	Elimination	40
2.6.1	Variable Elimination Algorithm	40
2.6.2	Linear-Gaussian Elimination	42
2.6.3	Sparse Cholesky Factor as a Bayes Net	45
2.7	Incremental SLAM	47
2.7.1	The Bayes Tree	48
2.7.2	Updating the Bayes Tree	49
2.7.3	Incremental Smoothing and Mapping	51
<b>3</b>	<b>Advanced State Variable Representations</b>	54
3.1	Optimization on Manifolds	54
3.1.1	Rotations and Poses	55
3.1.2	Matrix Lie Groups	56
3.1.3	Lie Group Optimization	57
3.1.4	Uncertainty and Lie Groups	59
3.1.5	Lie Group Extras	60
3.2	Continuous-Time Trajectories	62
3.2.1	Splines	63
3.2.2	From Parametric to Nonparametric	64
3.2.3	Gaussian Processes	66
3.2.4	Spline and GPs on Lie Groups	68
<b>4</b>	<b>Robustness to Incorrect Data Association and Outliers</b>	74
4.1	What Causes Outliers and Why Are They a Problem?	74
4.1.1	Data Association and Outliers	74
4.1.2	Least-Squares in the Presence of Outliers	76
4.2	Detecting and Rejecting Outliers in the SLAM Front-end	77
4.2.1	RANDom SAmple Consensus (RANSAC)	77
4.2.2	Graph-theoretic Outlier Rejection and Pairwise Consistency Maximization	80
4.3	Increasing Robustness to Outliers in the SLAM Back-end	84
4.3.1	Iteratively Reweighted Least Squares	88
4.3.2	Black-Rangarajan Duality	89
4.3.3	Alternating Minimization	91
4.3.4	Graduated Non-Convexity	92
4.4	Further References and New Trends	97

<b>5</b>	<b>Differentiable Optimization</b>	100
5.1	Introduction	100
5.1.1	Recap on Nonlinear Least Squares	101
5.2	Differentiation Through Nonlinear Least Squares	102
5.2.1	Unrolled Differentiation	103
5.2.2	Truncated Unrolled Differentiation	105
5.2.3	Implicit Differentiation	106
5.3	Differentiation on Manifold	108
5.3.1	Derivatives on the Lie Group	108
5.3.2	Differentiation Operations on Manifold	110
5.4	Modern Libraries	112
5.4.1	Numerical Challenges of Automatic Differentiation	112
5.4.2	Implementation of Differentiable Optimization	114
5.4.3	Related Open-source Libraries	115
5.5	Final Considerations & Recent Trends	117
<b>6</b>	<b>Dense Map Representations</b>	118
6.1	Range Sensing Preliminaries	118
6.1.1	Sensor Measurement Model	119
6.1.2	Conversion to Point Cloud	120
6.2	Foundations of Mapping	121
6.2.1	Occupancy Maps	122
6.2.2	Distance Fields	123
6.2.3	Occupancy Maps or Distance Fields?	125
6.3	Map Representations	125
6.3.1	Explicitness of Target Spatial Structures	125
6.3.2	Types of Spatial Abstractions	126
6.3.3	Data Structures and Storage	131
6.4	Constructing Maps: Methods and Practices	134
6.4.1	Points	134
6.4.2	Surfels	135
6.4.3	Meshes	135
6.4.4	Voxels	136
6.4.5	GPs	140
6.4.6	Hilbert Maps	142
6.4.7	Deep Learning in Mapping	143
6.5	Usage Considerations	143
6.5.1	Environmental Aspects	144
6.5.2	Downstream Task Types	145
6.5.3	Summary of Mapping Methods	146
<b>7</b>	<b>Certifiably Optimal Solvers and Theoretical Properties of SLAM</b>	148

<b>PART TWO SLAM IN PRACTICE</b>	149
<b>8 Prelude</b>	151
8.1 Structure of SLAM Framework	151
8.1.1 Odometry	152
8.1.2 Loop-closure	152
8.1.3 Priors and Unary Factors	153
8.2 Sensors in a Factor Graph	153
8.2.1 Selecting the Right Sensor for Your Application	154
8.2.2 Sensor Fusion	155
8.2.3 Calibration and Synchronization of Sensors	156
8.3 How to Read this Part?	157
<b>9 Visual SLAM</b>	158
9.1 Historic Background and Terminology	158
9.1.1 From Photogrammetry to Bundle Adjustment and Visual SLAM	158
9.1.2 Terminology	159
9.2 Visual SLAM Fundamentals	160
9.2.1 Camera Model	160
9.2.2 Keypoints	163
9.2.3 Reprojection Error	168
9.2.4 Keypoint-Based Visual SLAM	169
9.2.5 Photometric Error and Direct Methods	171
9.2.6 Visual Place Recognition and Global Localization	171
9.2.7 Initialization	171
9.2.8 Common Steps	172
9.2.9 Map Representations	172
9.3 The Processing Pipeline of a Visual SLAM System	172
9.3.1 Visual Odometry Front-End	172
9.3.2 Mapping Back-End	173
9.3.3 Visual Place Recognition and Relocalization	173
9.3.4 Compute and Data Flow	173
9.3.5 Keypoint-based Image Alignment	174
9.3.6 Direct Image Alignment	176
9.3.7 Solving BA	178
9.3.8 Examples of Full Visual SLAM Systems	180
9.4 Realtime Dense Reconstruction	181
9.5 SLAM with Depth-sensing Cameras	182
9.6 Combining Vision with Other Modalities	184
9.6.1 Inertial Measurement Units (IMU)	184
9.6.2 GPS and WiFi for Global Localization	186
9.7 Bundle Adjustment Revisited	187

	<i>Contents</i>	vii
9.8	Recent Developments	187
<b>10</b>	<b>LiDAR SLAM</b>	189
10.1	LiDAR Sensing Preliminary and Categorization	189
10.2	LiDAR Odometry	192
10.2.1	Foundations of Scan Registration	192
10.2.2	Common Components for LiDAR Odometry	195
10.2.3	Summary	200
10.3	LiDAR Place Recognition	201
10.3.1	Problem Definition	201
10.3.2	Methods for LiDAR Place Recognition	202
10.3.3	Summary	204
10.4	LiDAR SLAM	204
10.4.1	Structure of a LiDAR SLAM System	205
10.4.2	Pose-graph Optimization and Map Update	207
10.4.3	Multi-robot and Multi-session LiDAR SLAM	208
10.5	Outlook and Futures Challenges	212
<b>11</b>	<b>Radar SLAM</b>	215
11.1	Introduction to Radar	215
11.1.1	Sensor Types	215
11.1.2	Radar Sensing Principles	217
11.1.3	Challenges to Radar Applications	222
11.1.4	Radar Filtering	224
11.2	Radar Odometry	226
11.2.1	Doppler Odometry	226
11.2.2	Direct Odometry	229
11.2.3	Feature-based Odometry	230
11.2.4	Registration-based Odometry	230
11.2.5	Motion Compensation	232
11.3	Radar Place Recognition	233
11.3.1	Unique Challenges in Radar Place Recognition	233
11.3.2	Learning-based Radar PR	234
11.3.3	Descriptor-based Radar PR	236
11.4	Radar SLAM	237
11.4.1	Map Representations	237
11.4.2	Radar SLAM	238
11.4.3	Multi-modality in Radar SLAM	242
11.5	Radar Datasets	243
11.6	Outlook and challenges	245
<b>12</b>	<b>Event-based SLAM</b>	246
12.1	Sensor Description	246
12.1.1	Working principle	246

12.1.2	Advantages of Event Cameras	248
12.1.3	Current Devices and Trends.	249
12.2	Challenges and Applications	250
12.3	Methodology Overview	251
12.4	Front-end	253
12.4.1	Pre-processing. Event Representations	253
12.4.2	Indirect Methods	254
12.4.3	Direct Methods	255
12.4.4	Model-based and Learning-based Methods	256
12.5	Back-end	256
12.6	State-of-the-Art Systems	257
12.7	Datasets, Simulators, and Benchmarks	259
12.7.1	Simulators	259
12.7.2	Datasets and Benchmarks	261
12.7.3	Metrics	265
12.8	Outlook	266
13	<b>Inertial Odometry for SLAM</b>	267
13.1	Basics of Inertial Sensing and Navigation	267
13.1.1	Sensing Principles and Measurement Models	268
13.1.2	Initial Alignment	269
13.2	IMU Preintegration and Factor Graphs	270
13.2.1	Motion Integration	271
13.2.2	IMU Preintegration on Manifold	273
13.2.3	Advanced Preintegration Techniques	279
13.3	Observability of Aided Inertial Navigation	282
13.3.1	Linearized Measurement Models	283
13.3.2	Observability Analysis	286
13.3.3	Degenerate Motions	288
13.4	Visual-Inertial Odometry and Practical Considerations	288
13.4.1	Visual-Inertial Odometry	289
13.4.2	Extrinsic Calibration	292
13.4.3	Temporal Synchronization	292
13.5	New trends	293
14	<b>Leg Odometry for SLAM</b>	295
14.1	Introduction	295
14.1.1	Historical Background	296
14.1.2	Reference Frames	297
14.1.3	State Definition	298
14.1.4	Legged Robot Kinematics	298
14.1.5	Legged Robot Dynamics	300
14.1.6	Joint Sensing	301

	<i>Contents</i>	ix
14.2	Motion Estimation	304
14.2.1	Relative Pose Estimation	304
14.2.2	Velocity Estimation	306
14.3	Contact Estimation	307
14.3.1	With Contact Sensors	308
14.3.2	With Force/Torque Sensors	308
14.3.3	With IMUs	309
14.3.4	From Joint Torque Sensing	309
14.4	Leg Odometry for Estimation Problems	310
14.4.1	Encoder Noise Propagation	310
14.4.2	Factor Graph Smoothing	311
14.4.3	Integration with Exteroceptive Sensors for SLAM	314
14.5	Open Challenges	315
14.5.1	Leg Deformation	315
14.5.2	Non-rigid Contacts and Slippage	316
14.6	New Trends and Paradigm Shifts	317
14.6.1	Learning-based Contact Estimation	317
14.6.2	End-to-End Learning	317
14.6.3	Humanoid Robots	318
 <b>PART THREE FROM SLAM TO SPATIAL AI</b>		321
15	<b>Prelude</b>	323
16	<b>Spatial AI</b>	324
17	<b>Boosting SLAM with Deep Learning</b>	325
18	<b>NeRF and 3D Gaussian Splatting: Map Representations with Differentiable Volume Rendering</b>	326
19	<b>Dynamic and Deformable SLAM</b>	327
20	<b>Metric-Semantic SLAM</b>	328
21	<b>Foundation Models for SLAM</b>	329
 <i>Notes</i>		331
<i>References</i>		332
<i>Author index</i>		389
<i>Subject index</i>		390



## 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
$X$	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 $\mathbf{b}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	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) $\mathcal{F}^b$ : $\mathbf{v}^b = \mathbf{R}_a^b \mathbf{v}^a$
$\tilde{\mathbf{v}}_a^a = \begin{bmatrix} \mathbf{v}^a \\ t_a^b \end{bmatrix}$	The three-dimensional position of the origin of frame $\mathcal{F}^a$ expressed in $\mathcal{F}^b$
$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & t_a^b \\ \mathbf{0} & 1 \end{bmatrix}$	A $4 \times 1$ homogeneous point expressed in $\mathcal{F}^a$
SO(3)	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 rotations
SE(3)	The Lie algebra associated with SO(3)
SE(3)	The special Euclidean group, a matrix Lie group used to represent 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 rotations (resp. 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 rotations (resp. poses) to a vector in $\mathbb{R}^3$ (resp. $\mathbb{R}^6$ )

# **PART ONE**

## FOUNDATIONS OF SLAM



# 1

## Prelude

Luca Carlone, Ayoung Kim, Frank Dellaert, Timothy Barfoot, 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.

### 1.1 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. These type of representations 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, and 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. Figure 1.1 provides some real-world examples of simultaneous localization and mapping (SLAM) in action.

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 Simultaneous Localization and Mapping (SLAM). 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

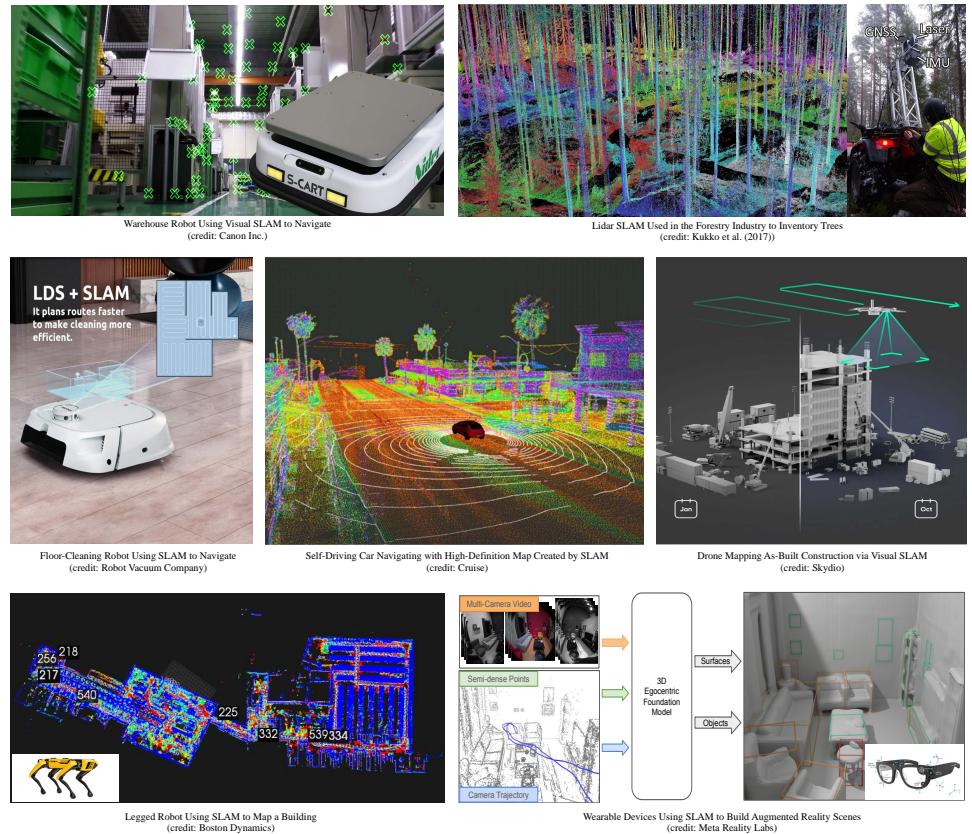


Figure 1.1 SLAM is rapidly becoming an enabling technology in a wide array of applications including warehouse robotics, forest inventories [407], floor-cleaning, self-driving cars, drones surveillance, legged-robot mapping, and augmented reality to name only a few.

positioning system is used (e.g., differential GPS or motion capture), 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.

## 1.2 Anatomy of a Modern SLAM System

The ultimate goal of SLAM is to infer a map representation and robot poses (*i.e.*, 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)s, radars). In mathematical terms this can be understood as an *inverse problem*: given a set of measurements, 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. Instead of using every pixel in an image, these 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 10 and Chapter 11. Both indirect and direct methods have their advantages and shortcomings.

Indirect methods are often faster and more memory efficient. Rather than processing every single pixel of each camera image, for example, 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 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 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 overall loss function, 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 2 and Chapter 3— that have their foundations in probabilistic inference and that rephrase estimation in terms of solving optimization problems.

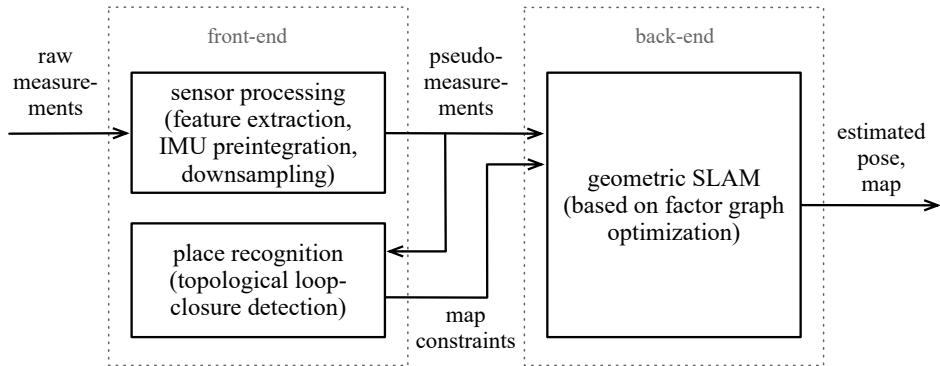


Figure 1.2 The anatomy of a typical SLAM is made up of a *front-end* (to process rich sensing data into more manageable information and to detect topological loop closures) and a *back-end* (to estimate the robot’s pose and a geometric map). The back-end often has a number of helper modules aimed at helping with robustness, computational tractability, and map quality.

Indirect methods produce a natural split in common SLAM architectures (Figure 1.2): 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.

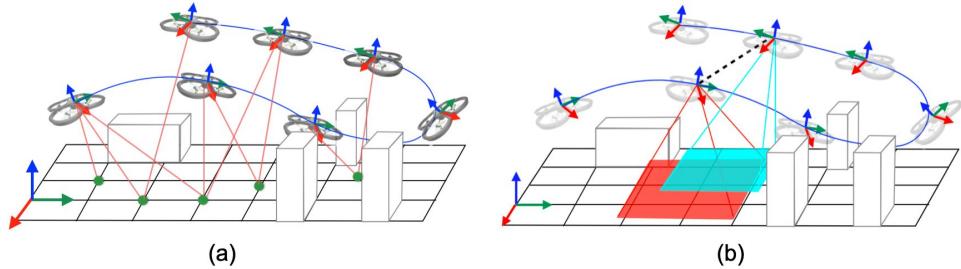


Figure 1.3 (a) In landmark-based SLAM models, the front-end produces measurements to 3D landmarks and the back-end estimates the robot trajectory (as a set of poses) and landmark positions. (b) In pose-graph-based SLAM models, the front-end abstracts the raw sensor measurements in terms of odometry and loop closure measurements (these are typically relative pose measurements) and the back-end estimates the overall robot trajectory.

**Example 1.1** (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 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 will also compute 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 1.3(a). We will discuss visual SLAM at length in Chapter 9.

**Example 1.2** (Lidar SLAM: from scans to odometry and loop closures). Lidar

<sup>1</sup> A more subtle point is that minimal solvers will 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 4.

SLAM uses lidar scans to estimate the robot trajectory and a map. A common 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*. This example leads to a pose-graph-based SLAM model, visualized in Figure 1.3(b). We discuss LiDAR SLAM in Chapter 10.

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 1.2): 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 bundle adjustment); 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.*, bundle adjustment 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.

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.

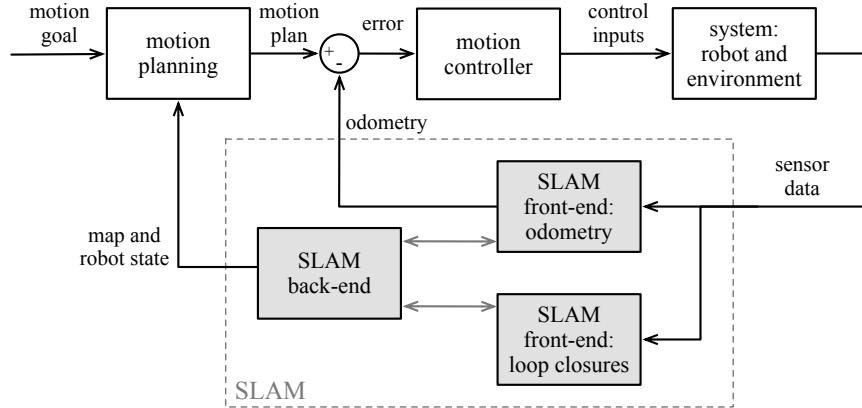


Figure 1.4 SLAM plays an important role in the overall autonomy pipeline of a robot that interacts with the world, and provides necessary information for control and motion planning.

### 1.3 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 1.4). 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 1.4, 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 1.4) 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). We

also observe that the processes involved in a SLAM system have complex interactions (as emphasized by the bi-directional edges in Figure 1.4): 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.

The problem of visual SLAM is closely related to the problem of Structure from Motion (SfM). While for some researchers both terms are equivalent, others distinguish and argue that visual SLAM systems will typically also integrate additional sensory information (IMUs, wheel odometry, etc) and focus on an online approach where data comes in sequentially whereas in SfM both online and offline versions are conceivable and the input is only images.

Overall, one can distinguish two complementary challenges: There is the *online challenge*, where a robot moves around, where sensory data streams in sequentially and while the SLAM back-end might run at a slower pace, vital estimates such as the localization of the robot must be determined in real-time, often even on embedded hardware with limited compute. These realtime constraints are vital for the robot to properly act in a complex environment, in particular with faster robots such as drones. They often dictate the choice of algorithms and processing steps.

And there is the *offline challenge* where the input data may not exhibit any sequential ordering (say an unordered dataset of images), where computations typically do not require real-time performance and where the compute hardware can be (arbitrarily) large (multiple powerful GPUs), see for example [17]. In such cases, accuracy of the estimated map and trajectories is more important than compute time.

In most applications, however, one will face a mix of these two extreme scenarios where certain quantities need to be determined fast whereas others can be determined offline. In practical applications of SLAM it is of utmost importance to carefully analyze which quantities need to be determined at which frequency and one may come up with an entire hierarchy of different temporal scales at which quantities are being estimated.

### **1.3.1 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.

***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 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.

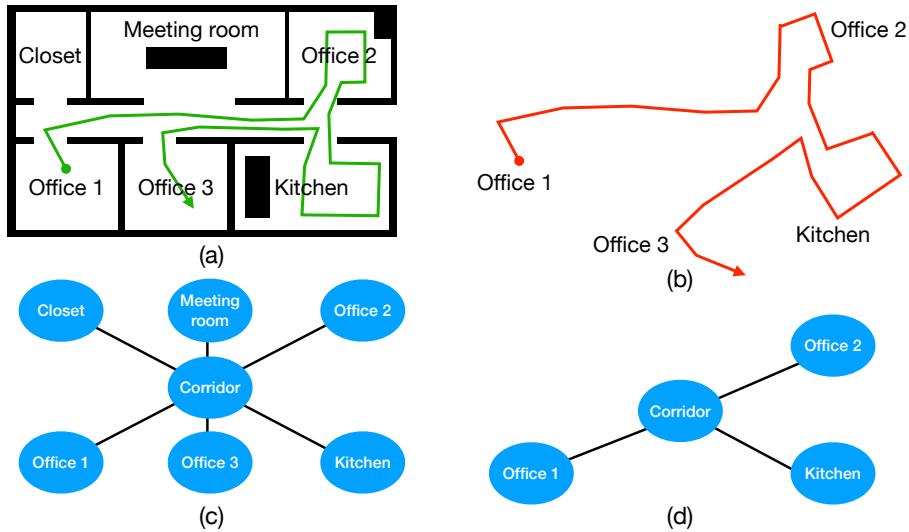


Figure 1.5 (a) Our robot visits Office 1 in a building and then —after exploring other areas (including Office 2 and the Kitchen)— it visits Office 3, which is just next door from Office 1. Obstacles are shown in black and ground truth trajectory is shown in green. (b) Odometric estimate of the trajectory, 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

<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.

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 1.5(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 1.5(b)). A slightly more sophisticated alternative is to 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 1.5(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 1.5(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 4.

**Q3. Do we need maps?** SLAM builds a map that can be directly queried, inspected, and visualized. As we will see in Chapter 6, 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 [591, 809]. 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.

## 1.4 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.

### 1.4.1 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 largest mountain named after him. In 1856, Carl Maximilian von Bauernfeind published a standard book on “Elements of Surveying” [46]. 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 in computer vision. Its origins can be traced back to the 19th century. See Chapter 9.

In robotics, the origin of SLAM is typically traced back to the seminal work of Smith and Chessman [656] and Durrant-Whyte [192], as well as the parallel work by Crowley [151] and Chatila and Laumond [114]. The acronym SLAM was coined in 1995, as part of the survey paper [193]. 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,

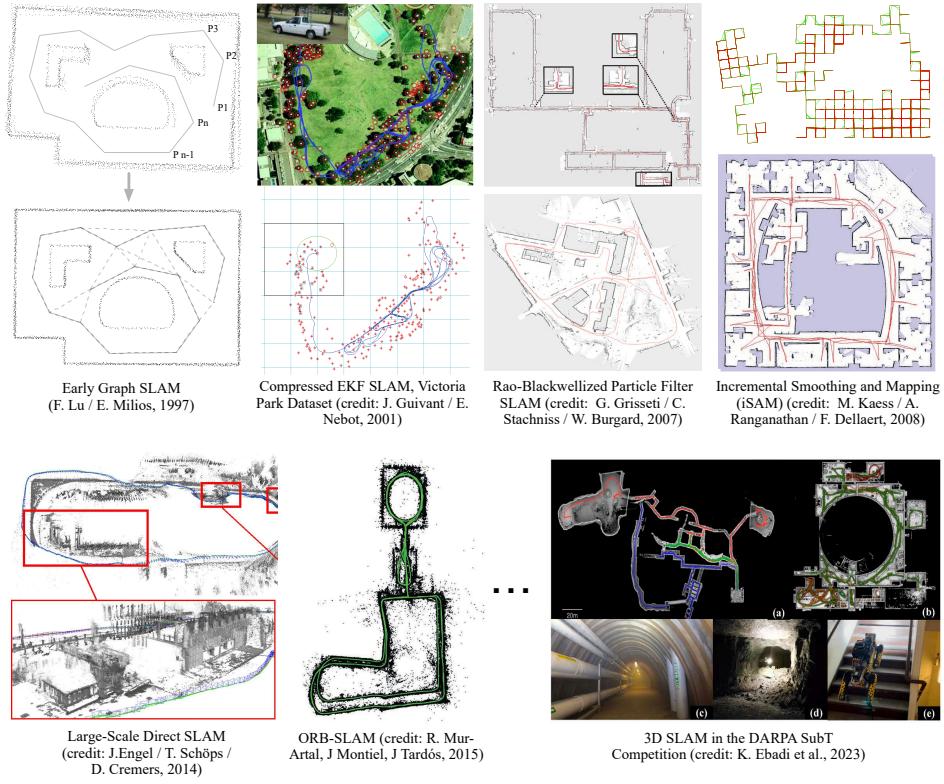


Figure 1.6 The history of SLAM is filled with numerous advances that have led to modern SLAM systems capable of localizing and mapping robots in challenging real-world environments. This image shows a selection of representative highlights.

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. 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* [504, 655, 269], 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 robot poses.

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 [210]; 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 [464], 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 [160, 353].<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 7). 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

<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 2.

Part III of this handbook), where one wants to estimate both continuous variables (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 [508]), and novel estimation formulations have been developed, including invariant [44] and equivariant filters [223], as well as alternative formulations based on random finite sets [511].

#### **1.4.2 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

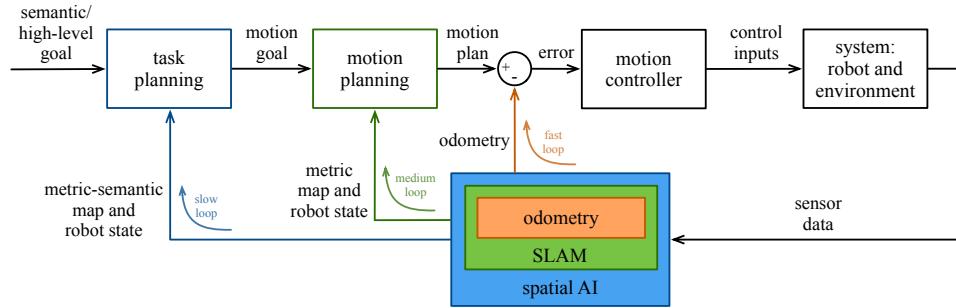


Figure 1.7 Spatial AI (or spatial perception) extends the geometric reasoning capabilities of SLAM to also perform semantic and physical reasoning. While the SLAM block is informed by odometry and provides a geometric understanding of the scene, the Spatial AI block is informed by the SLAM results and 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.

think that Spatial AI has SLAM as a submodule (to handle the geometric reasoning part), but provides extra semantic reasoning capabilities. This allows closing the loop over task planning, as shown in Figure 1.7, 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.

## 1.5 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 2 introduces the factor-graph formulation of SLAM and reviews how to solve it via iterative nonlinear optimization methods. Then, Chapter 3 takes the indispensable step of extending the formulation to allow the estimation of variables belonging to smooth manifolds, such as rotation and poses. Chapter 4 discusses how to model and mitigate the impact of outliers and incorrect data association in the SLAM back-end. Chapter 5 reviews techniques to make the back-end optimization differentiable, a key step towards interfacing traditional SLAM methods with more recent deep learning architectures. Chapter 6 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 7 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 9 reviews the large body of literature on visual SLAM. Chapter 10 and Chapter 11 cover lidar-SLAM and radar-SLAM, respectively. Chapter 12 discusses recent work on SLAM using event-based cameras. Chapter 13 reviews how to model inertial measurements as part of a factor-graph SLAM system and discusses fundamental limits (*e.g.*, observability). Chapter 14 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 16 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. Chapter 17 reviews recent improvements obtained by introducing deep learning modules in conjunction with differentiable optimization in SLAM. Chapter 18 discusses opportunities and challenges in using novel map presentations, including neural radiance fields (NeRFs) and Gaussian Splatting. Chapter 19 covers recent work on SLAM in highly dynamic and deformable environments, touching on real applications from mapping in crowded environments to surgical robotics. Chapter 20 discusses progress in Spatial AI and metric-semantic map representations. Finally, Chapter 21 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.

## 2

# Factor Graphs for SLAM

Frank Dellaert, Michael Kaess, 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.

### *Historical Note*

A *smoothing* approach to SLAM involves not just the most current robot location, but the entire robot trajectory up to the current time. A number of authors consider the problem of smoothing the robot trajectory only [114, 462, 463, 283, 395, 209], now known as *PoseSLAM*. This is particularly suited to sensors such as laser-range finders that yield pairwise constraints between nearby robot poses.

More generally, one can consider the *full SLAM problem* [697], i.e., the 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 [188, 230, 229, 697]. 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 [558, 696], 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 [160, 163] 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

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 [59]. The use of square roots results in more accurate and stable algorithms, and, quoting Maybeck [490], “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”.

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. [164].

## 2.1 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.

### 2.1.1 A Toy Example

We begin by examining a simple SLAM scenario to illustrate how factor graphs are constructed. Figure 2.1 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

$$\int p(\mathbf{x}) d\mathbf{x} = 1, \quad (2.1)$$

which is the axiom of total probability. In the simple example of Figure 2.1, 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.

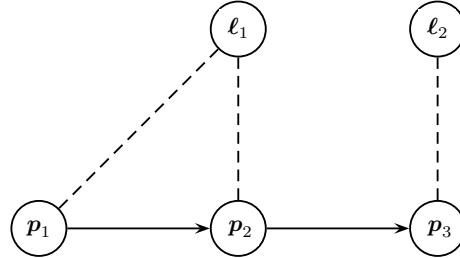


Figure 2.1 A toy simultaneous localization and mapping (SLAM) example with three robot poses and two landmarks. Above we schematically indicate the robot motion with arrows, while the dotted lines indicate bearing measurements.

state,  $\mathbf{x}$ , is

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \ell_1 \\ \ell_2 \end{bmatrix}, \quad (2.2)$$

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

$$p(\mathbf{x}|\mathbf{z}), \quad (2.3)$$

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 [394]. 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.

### 2.1.2 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.

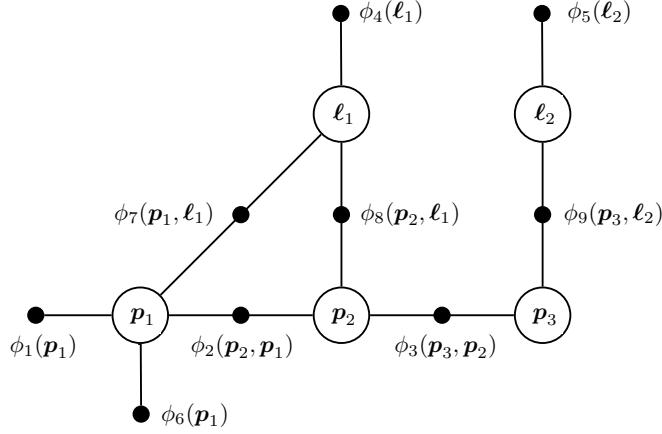


Figure 2.2 Factor graph resulting from the example in Figure 2.1.

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

$$p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{p}_1) p(\mathbf{p}_2|\mathbf{p}_1) p(\mathbf{p}_3|\mathbf{p}_2) \quad (2.4a)$$

$$\times p(\ell_1) p(\ell_2) \quad (2.4b)$$

$$\times p(\mathbf{z}_1|\mathbf{p}_1) \quad (2.4c)$$

$$\times p(\mathbf{z}_2|\mathbf{p}_1, \ell_1) p(\mathbf{z}_3|\mathbf{p}_2, \ell_1) p(\mathbf{z}_4|\mathbf{p}_3, \ell_2). \quad (2.4d)$$

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,  $\mathbf{x}$ .

To visualize this factorization, we use a *factor graph*. Figure 2.2 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

$$\phi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ell_1, \ell_2) = \phi_1(\mathbf{p}_1) \phi_2(\mathbf{p}_2, \mathbf{p}_1) \phi_3(\mathbf{p}_3, \mathbf{p}_2) \quad (2.5a)$$

$$\times \phi_4(\ell_1) \phi_5(\ell_2) \quad (2.5b)$$

$$\times \phi_6(\mathbf{p}_1) \quad (2.5c)$$

$$\times \phi_7(\mathbf{p}_1, \ell_1) \phi_8(\mathbf{p}_2, \ell_1) \phi_9(\mathbf{p}_3, \ell_2), \quad (2.5d)$$

where the correspondence between the factors and the original probability densities in (2.4a)-(2.4d) 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  $\phi_{\mathbf{z}_4}(\mathbf{p}_3, \ell_2)$ .

### 2.1.3 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 2.1 is an example of *landmark-based SLAM* because it involves both pose and landmark variables. Figure 2.3 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 2.4. 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,

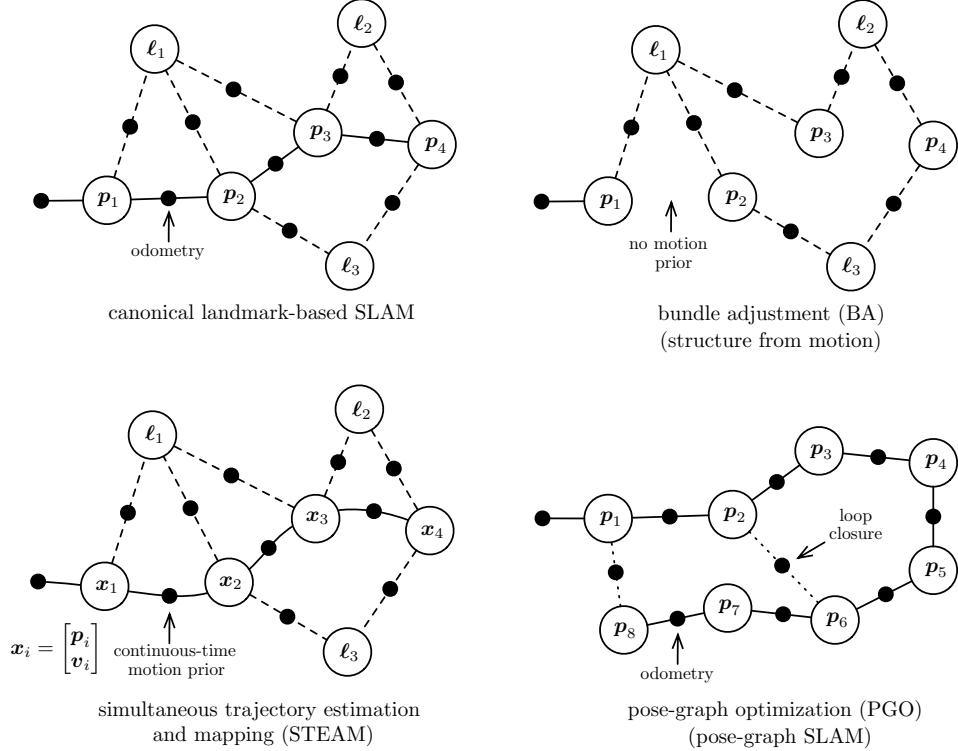


Figure 2.3 A few variants of SLAM problems that can all be viewed through the factor-graph lens. Canonical landmark-based SLAM has both pose and landmark variables; landmarks are measured from poses and there is some motion prior between poses typically based on odometry. BA is the same but without the 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.

## 2.2 From MAP Inference to Least Squares

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

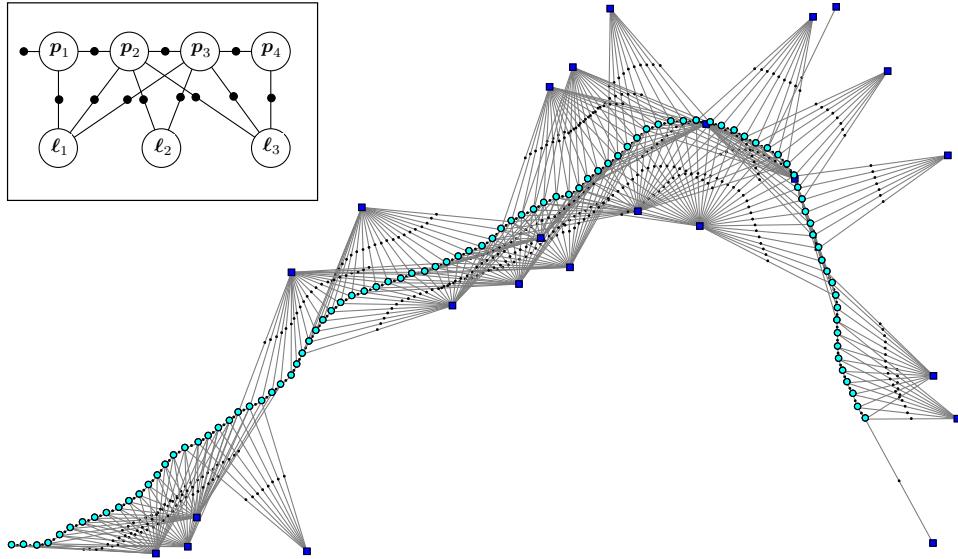


Figure 2.4 Factor graph for a larger, simulated SLAM example.

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.

### 2.2.1 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  $\mathbf{z}$ :

$$\mathbf{x}^{\text{MAP}} = \arg \max_{\mathbf{x}} p(\mathbf{x}|\mathbf{z}) \quad (2.6a)$$

$$= \arg \max_{\mathbf{x}} \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z})} \quad (2.6b)$$

$$= \arg \max_{\mathbf{x}} p(\mathbf{z}|\mathbf{x})p(\mathbf{x}) \quad (2.6c)$$

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

$$\phi(\mathbf{x}) = \prod_i \phi_i(\mathbf{x}_i). \quad (2.7)$$

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 (2.7) of all factor-graph potentials:

$$\mathbf{x}^{\text{MAP}} = \arg \max_{\mathbf{x}} \phi(\mathbf{x}) \quad (2.8a)$$

$$= \arg \max_{\mathbf{x}} \prod_i \phi_i(\mathbf{x}_i). \quad (2.8b)$$

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.

### 2.2.2 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

$$\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\Sigma}}^2\right), \quad (2.9)$$

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

$$\|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\Sigma}}^2 \triangleq (\boldsymbol{\theta} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}) \quad (2.10)$$

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

$$\mathbf{z} = \mathbf{h}(\mathbf{p}, \ell) + \boldsymbol{\eta}, \quad (2.11)$$

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  $\mathbf{z}$ :

$$p(\mathbf{z}|\mathbf{p}, \ell) = \mathcal{N}(\mathbf{z}; \mathbf{h}(\mathbf{p}, \ell), \Sigma_R) = \frac{1}{\sqrt{|2\pi\Sigma_R|}} \exp\left(-\frac{1}{2}\|\mathbf{z} - \mathbf{h}(\mathbf{p}, \ell)\|_{\Sigma_R}^2\right). \quad (2.12)$$

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

$$\mathbf{h}(\mathbf{p}, \ell) = \text{atan2}(\ell_y - p_y, \ell_x - p_x), \quad (2.13)$$

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

$$p(\mathbf{z}|\mathbf{p}, \ell) = \frac{1}{\sqrt{|2\pi\Sigma_R|}} \exp\left(-\frac{1}{2}\|\mathbf{z} - \text{atan2}(\ell_y - p_y, \ell_x - p_x)\|_{\Sigma_R}^2\right). \quad (2.14)$$

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 4.

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* that the robot is assumed to obey given known control inputs  $\mathbf{u}_t$ . In practice, we often use a conditional Gaussian assumption,

$$p(\mathbf{p}_{t+1}|\mathbf{p}_t, \mathbf{u}_t) = \frac{1}{\sqrt{|2\pi\Sigma_Q|}} \exp\left(-\frac{1}{2}\|\mathbf{p}_{t+1} - \mathbf{g}(\mathbf{p}_t, \mathbf{u}_t)\|_{\Sigma_Q}^2\right), \quad (2.15)$$

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

$$p(\mathbf{o}_t|\mathbf{p}_{t+1}, \mathbf{p}_t) = \frac{1}{\sqrt{|2\pi\Sigma_S|}} \exp\left(-\frac{1}{2}\|\mathbf{o}_t - (\mathbf{p}_{t+1} - \mathbf{p}_t)\|_{\Sigma_S}^2\right). \quad (2.16)$$

If we have *both* known control inputs  $\mathbf{u}_t$  and odometry measurements  $\mathbf{o}_t$  we can combine (2.15) and (2.16).

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.

### 2.2.3 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

$$\phi_i(\mathbf{x}_i) \propto \exp\left(-\frac{1}{2} \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2\right), \quad (2.17)$$

which include both simple Gaussian priors and likelihood factors derived from measurements corrupted by zero-mean, normally distributed noise. Taking the negative log of (2.8b) and dropping the factor  $\frac{1}{2}$  allows us to instead minimize a sum of *nonlinear least-squares* terms:

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2. \quad (2.18)$$

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 (2.18) 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 (2.18). We should caution, however, that as our objective in (2.18) 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.

## 2.3 Solving Linear Least Squares

Before tackling the more difficult problem of nonlinear least squares, in this section we first show to *linearize* the problem, show how this leads to a *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 book by Golub and Loan [265].

### 2.3.1 Linearization

We can linearize all measurement functions  $\mathbf{h}_i(\cdot)$  in the nonlinear least-squares objective function (2.18) using a simple Taylor expansion,

$$\mathbf{h}_i(\mathbf{x}_i) = \mathbf{h}_i(\mathbf{x}_i^0 + \boldsymbol{\delta}_i) \approx \mathbf{h}_i(\mathbf{x}_i^0) + \mathbf{H}_i \boldsymbol{\delta}_i, \quad (2.19)$$

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

$$\mathbf{H}_i \triangleq \left. \frac{\partial \mathbf{h}_i(\mathbf{x}_i)}{\partial \mathbf{x}_i} \right|_{\mathbf{x}_i^0}, \quad (2.20)$$

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 3.

Substituting the Taylor expansion (2.19) into the nonlinear least-squares expression (2.18) we obtain a *linear* least-squares problem in the state update vector  $\boldsymbol{\delta}$ ,

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0) - \mathbf{H}_i \boldsymbol{\delta}_i\|_{\Sigma_i}^2 \quad (2.21a)$$

$$= \arg \min_{\boldsymbol{\delta}} \sum_i \|(\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)) - \mathbf{H}_i \boldsymbol{\delta}_i\|_{\Sigma_i}^2, \quad (2.21b)$$

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:

$$\|e\|_{\Sigma}^2 \triangleq e^\top \Sigma^{-1} e = \left( \Sigma^{-1/2} e \right)^\top \left( \Sigma^{-1/2} e \right) = \left\| \Sigma^{-1/2} e \right\|_2^2. \quad (2.22)$$

Hence, we can eliminate the covariances  $\Sigma_i$  by pre-multiplying the Jacobian  $\mathbf{H}_i$  and the prediction error in each term in (2.21b) with  $\Sigma_i^{-1/2}$ :

$$\mathbf{A}_i = \Sigma_i^{-1/2} \mathbf{H}_i \quad (2.23a)$$

$$\mathbf{b}_i = \Sigma_i^{-1/2} (\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)). \quad (2.23b)$$

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.

### 2.3.2 SLAM as Least-Squares

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

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{A}_i \boldsymbol{\delta}_i - \mathbf{b}_i\|_2^2 \quad (2.24a)$$

$$= \arg \min_{\boldsymbol{\delta}} \|\mathbf{A} \boldsymbol{\delta} - \mathbf{b}\|_2^2, \quad (2.24b)$$

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 the classical linear algebra approach to solving this least-squares problem.

### 2.3.3 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 (2.24b) can be found by solving the *normal equations*:

$$(\mathbf{A}^\top \mathbf{A}) \boldsymbol{\delta}^* = \mathbf{A}^\top \mathbf{b}. \quad (2.25)$$

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

$$\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A} = \mathbf{R}^\top \mathbf{R}. \quad (2.26)$$

Above, the *Cholesky triangle*  $\mathbf{R}$  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

$$\mathbf{R}^\top \mathbf{y} = \mathbf{A}^\top \mathbf{b} \quad (2.27)$$

for  $\mathbf{y}$  and then

$$\mathbf{R} \boldsymbol{\delta}^* = \mathbf{y} \quad (2.28)$$

<sup>3</sup> Some treatments, including [265], define the Cholesky triangle as the lower-triangular matrix  $\mathbf{L} = \mathbf{R}^\top$ , but the other convention is more convenient here.

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 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:

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix} = \mathbf{Q}^\top \mathbf{b}. \quad (2.29)$$

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:

$$\mathbf{H}_n \cdots \mathbf{H}_2 \mathbf{H}_1 \mathbf{A} = \mathbf{Q}^\top \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}. \quad (2.30)$$

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

$$\|\mathbf{A} \boldsymbol{\delta} - \mathbf{b}\|_2^2 = \|\mathbf{Q}^\top \mathbf{A} \boldsymbol{\delta} - \mathbf{Q}^\top \mathbf{b}\|_2^2 = \|\mathbf{R} \boldsymbol{\delta} - \mathbf{d}\|_2^2 + \|\mathbf{e}\|_2^2, \quad (2.31)$$

where we made use of the equalities from (2.29). 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

$$\mathbf{R} \boldsymbol{\delta}^* = \mathbf{d} \quad (2.32)$$

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

$$\mathbf{A}^\top \mathbf{A} = \left[ \begin{array}{c} \mathbf{R} \\ \mathbf{0} \end{array} \right]^\top \mathbf{Q}^\top \mathbf{Q} \left[ \begin{array}{c} \mathbf{R} \\ \mathbf{0} \end{array} \right] = \mathbf{R}^\top \mathbf{R}, \quad (2.33)$$

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 [160, 163].

## 2.4 Nonlinear Optimization

In this section, we discuss some classic optimization approaches to the nonlinear least-squares problem defined in (2.18). As a reminder, in SLAM the nonlinear least-squares objective function is given by

$$J(\mathbf{x}) \triangleq \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2 \quad (2.34)$$

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 (2.18) in order to approach the minimum [170]. 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 [537], while [265] 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.

### 2.4.1 Steepest Descent

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

$$\boldsymbol{\delta}^{\text{sd}} = -\alpha \nabla J(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^t}. \quad (2.35)$$

Here the negative gradient is used to identify the direction of steepest descent. For the nonlinear least-squares objective function (2.34), 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.

### 2.4.2 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 (2.25),

$$\mathbf{A}^\top \mathbf{A} \boldsymbol{\delta}^{\text{gn}} = \mathbf{A}^\top \mathbf{b}, \quad (2.36)$$

by any of the methods in Section 2.3.3. 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.

### 2.4.3 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 [430] proposed to modify the normal equations (2.25) by adding a non-negative constant  $\lambda \in \mathbb{R}^+ \cup \{0\}$  to the diagonal

$$(\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I}) \boldsymbol{\delta}^{\text{lm}} = \mathbf{A}^\top \mathbf{b}. \quad (2.37)$$

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  $J$  (2.34). Hence, LM can be seen to blend naturally between the GN and SD methods.

Marquardt [484] later proposed to take into account the scaling of the diagonal entries to provide faster convergence:

$$(\mathbf{A}^\top \mathbf{A} + \lambda \text{diag}(\mathbf{A}^\top \mathbf{A})) \boldsymbol{\delta}^{\text{lm}} = \mathbf{A}^\top \mathbf{b}. \quad (2.38)$$

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.

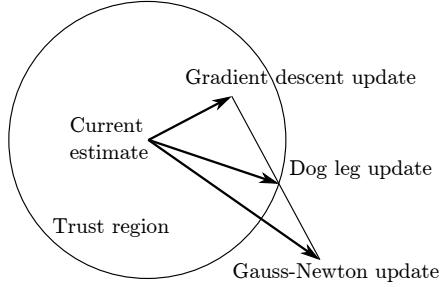


Figure 2.5 Powell’s dogleg algorithm combines the separately computed Gauss-Newton and gradient descent update steps.

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.

#### 2.4.4 Dogleg Minimization

Powell’s dogleg (PDL) algorithm [578] can be a more efficient alternative to LM [460]. 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 2.5 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

$$\rho = \frac{J(\mathbf{x}^t) - J(\mathbf{x}^t + \boldsymbol{\delta})}{L(\mathbf{0}) - L(\boldsymbol{\delta})}, \quad (2.39)$$

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 (2.34) 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.

## 2.5 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 2.1 a dense method will work fine. The larger simulation example, with its factor graph shown in Figure 2.4, 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 2.4 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.

### 2.5.1 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:

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \sum_i \|\mathbf{A}_i \boldsymbol{\delta}_i - \mathbf{b}_i\|_2^2. \quad (2.40)$$

Each term above is derived from a factor in the original, nonlinear SLAM problem, linearized around the current linearization point (2.21b). 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 2.1. After linearization, we obtain a sparse system  $[\mathbf{A}|\mathbf{b}]$  with the block structure in Figure 2.6. 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 2.6 Block structure of the sparse Jacobian  $\mathbf{A}$  for the toy SLAM example in Figure 2.1 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.

$$\begin{bmatrix} \mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{13} & \mathbf{\Lambda}_{14} & & \\ & \mathbf{\Lambda}_{22} & & & \mathbf{\Lambda}_{25} \\ \mathbf{\Lambda}_{31} & \mathbf{\Lambda}_{33} & \mathbf{\Lambda}_{34} & & \\ \mathbf{\Lambda}_{41} & \mathbf{\Lambda}_{43} & \mathbf{\Lambda}_{44} & \mathbf{\Lambda}_{45} & \\ & \mathbf{\Lambda}_{52} & \mathbf{\Lambda}_{54} & \mathbf{\Lambda}_{55} & \end{bmatrix}$$

Figure 2.7 The information matrix  $\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A}$  for the toy SLAM problem.

### 2.5.2 The Sparse Information Matrix and its Graph

When using Cholesky factorization for solving the normal equations, as explained in Section 2.3.3, 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.

The information matrix  $\mathbf{\Lambda}$  can be associated with another, *undirected* graphical model for the SLAM problem, namely a *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 2.7 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 2.8 shows the corresponding MRF.

In what follows, we will frequently refer to the undirected graph  $G$  of the MRF associated with an inference problem. However, we will not use the MRF graphical

<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.

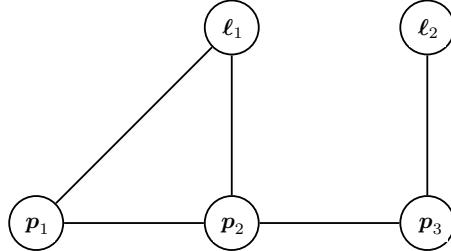


Figure 2.8 The Hessian matrix  $\Lambda$  can be interpreted as the matrix associated with the Markov random field representation for the problem.

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.

### 2.5.3 Sparse Factorization

We have seen MAP estimation amounts to solving a linear system of equations as described in Section 2.3.3. 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 [127] and SuiteSparseQR [154], 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 [798], 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 2.4. The sparsity patterns for the corresponding sparse Jacobian matrix  $\mathbf{A}$  is shown in Figure 2.9. 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 2.9, 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 [25, 155] 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 [747]. However, sparse factorization is still the method of choice for most SLAM problems and has a nice graphical model interpretation, which we discuss next.

## 2.6 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.

### 2.6.1 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).

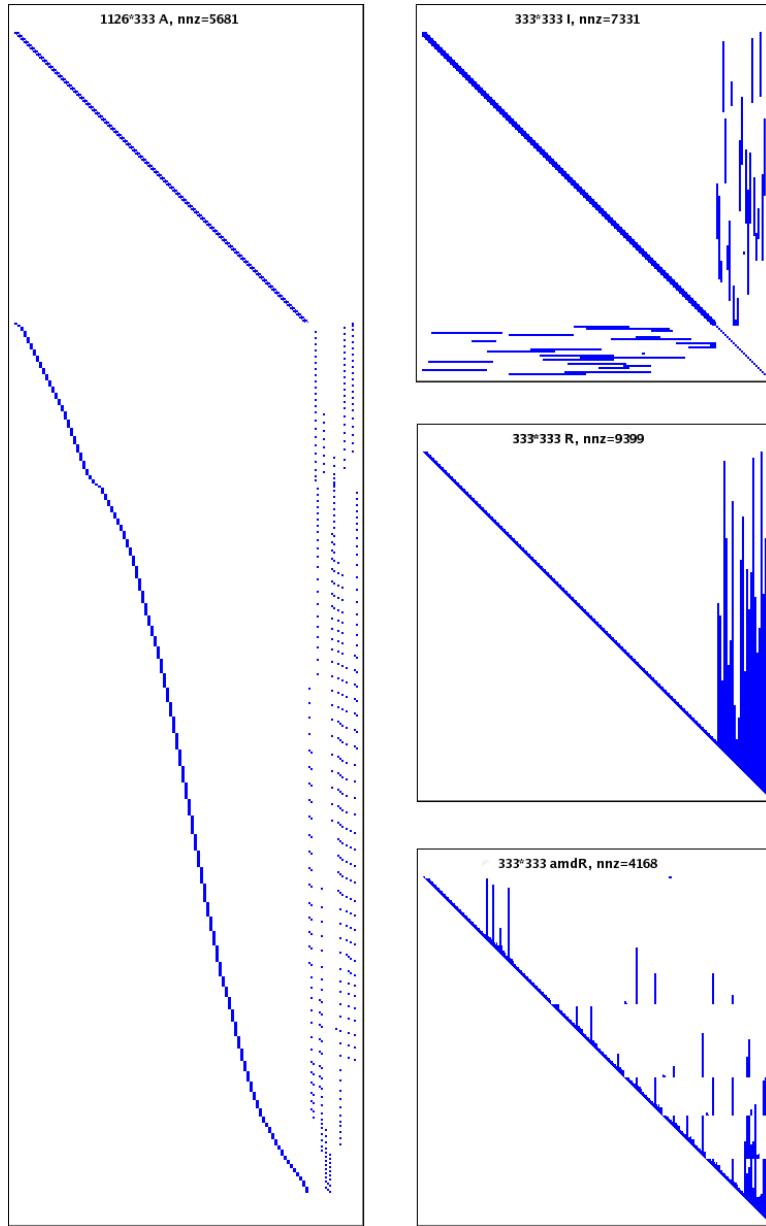


Figure 2.9 On the left, the measurement Jacobian  $\mathbf{A}$  associated with the problem in Figure 2.4, which has  $3 \times 95 + 2 \times 24 = 333$  unknowns. The number of rows, 1126, is equal to the number of (scalar) measurements. Also given is the number of nonzero entries “nnz”. On the right: (top) the information matrix  $\mathbf{\Lambda} \triangleq \mathbf{A}^\top \mathbf{A}$ ; (middle) its upper triangular Cholesky triangle  $\mathbf{R}$ ; (bottom) an alternative factor  $amdR$  obtained with a better variable ordering (COLAMD).

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

$$\phi(\mathbf{x}) = \phi(\mathbf{x}_1, \dots, \mathbf{x}_n) \quad (2.41)$$

into a factored Bayes net probability density of the form

$$p(\mathbf{x}) = p(\mathbf{x}_1|\mathbf{s}_1)p(\mathbf{x}_2|\mathbf{s}_2)\dots p(\mathbf{x}_n) = \prod_j p(\mathbf{x}_j|\mathbf{s}_j), \quad (2.42)$$

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  $\mathbf{s}_j$ :

$$\psi(\mathbf{x}_j, \mathbf{s}_j) = p(\mathbf{x}_j|\mathbf{s}_j)\tau(\mathbf{s}_j). \quad (2.43)$$

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 2.10, 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 2.10 (bottom right), corresponding to the factorization

$$p(\ell_1, \ell_2, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = p(\ell_1|\mathbf{p}_1, \mathbf{p}_2)p(\ell_2|\mathbf{p}_3)p(\mathbf{p}_1|\mathbf{p}_2)p(\mathbf{p}_2|\mathbf{p}_3)p(\mathbf{p}_3). \quad (2.44)$$

### 2.6.2 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

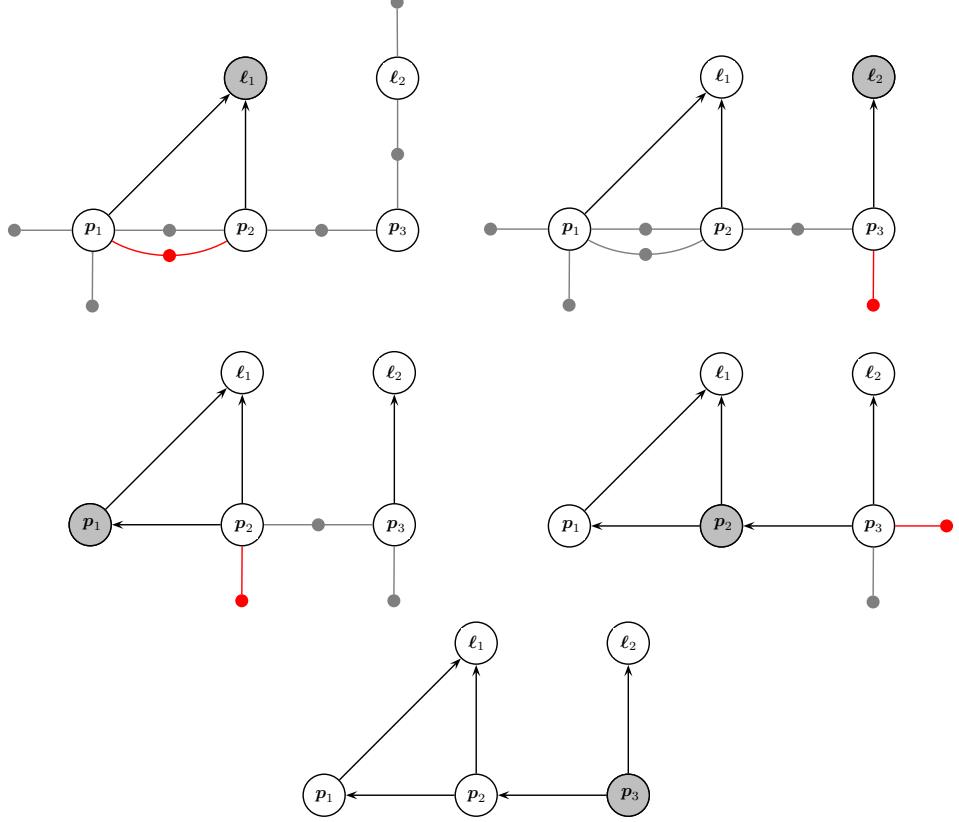


Figure 2.10 Variable elimination for the toy SLAM example, transforming the factor graph from Figure 2.2 into a Bayes net (bottom right), using the ordering  $\ell_1, \ell_2, p_1, p_2, p_3$ .

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

$$\psi(\mathbf{x}_j, \mathbf{s}_j) \leftarrow \prod_{i \in \mathcal{N}_j} \phi_i(\mathbf{x}_i) \quad (2.45a)$$

$$= \exp\left(-\frac{1}{2} \sum_i \|\mathbf{A}_i \mathbf{x}_i - \mathbf{b}_i\|_2^2\right) \quad (2.45b)$$

$$= \exp\left(-\frac{1}{2} \|\bar{\mathbf{A}}_j[\mathbf{x}_j; \mathbf{s}_j] - \bar{\mathbf{b}}_j\|_2^2\right), \quad (2.45c)$$

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

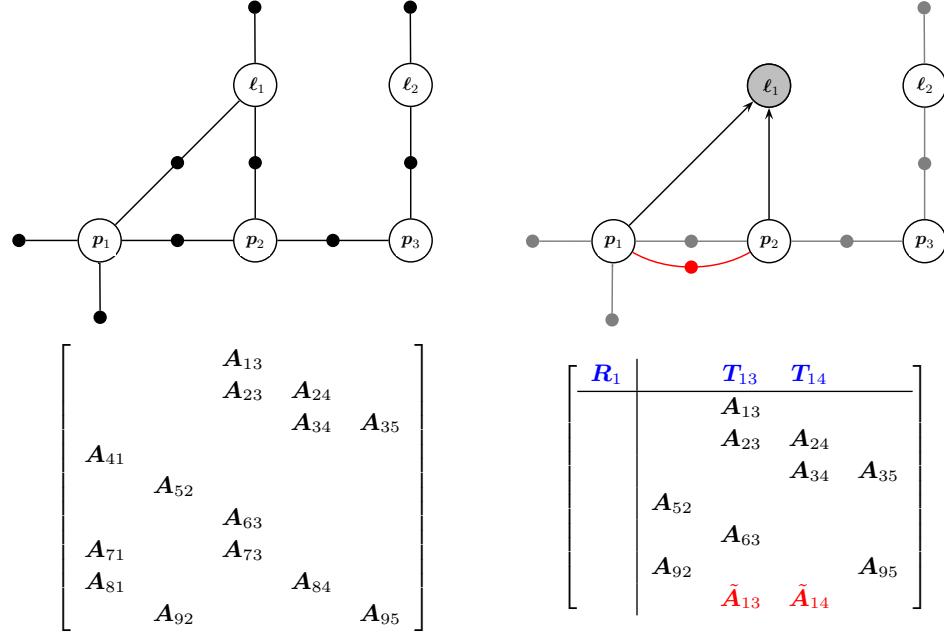


Figure 2.11 Eliminating the variable  $\ell_1$  as a partial sparse factorization step.

then equal to

$$\psi(\ell_1, p_1, p_2) = \exp\left(-\frac{1}{2}\|\bar{\mathbf{A}}_1[\ell_1; p_1; p_2] - \bar{\mathbf{b}}_1\|_2^2\right), \quad (2.46)$$

with

$$\bar{\mathbf{A}}_1 \triangleq \begin{bmatrix} \mathbf{A}_{41} & & \\ \mathbf{A}_{71} & \mathbf{A}_{73} & \\ \mathbf{A}_{81} & & \mathbf{A}_{84} \end{bmatrix}, \quad \bar{\mathbf{b}}_1 \triangleq \begin{bmatrix} \mathbf{b}_4 \\ \mathbf{b}_7 \\ \mathbf{b}_8 \end{bmatrix}. \quad (2.47)$$

Looking at the sparse Jacobian in Figure 2.6, 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 [265] as follows:

$$[\bar{\mathbf{A}}_j | \bar{\mathbf{b}}_j] = \mathbf{Q} \begin{bmatrix} \mathbf{R}_j & \mathbf{T}_j & \mathbf{d}_j \\ & \tilde{\mathbf{A}}_\tau & \tilde{\mathbf{b}}_\tau \end{bmatrix}, \quad (2.48)$$

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

$$\psi(\mathbf{x}_j, \mathbf{s}_j) = \exp \left\{ -\frac{1}{2} \|\bar{\mathbf{A}}_j[\mathbf{x}_j; \mathbf{s}_j] - \bar{\mathbf{b}}_j\|_2^2 \right\} \quad (2.49a)$$

$$= \exp \left\{ -\frac{1}{2} \|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2 \right\} \exp \left\{ -\frac{1}{2} \|\tilde{\mathbf{A}}_\tau \mathbf{s}_j - \tilde{\mathbf{b}}_\tau\|_2^2 \right\} \\ = p(\mathbf{x}_j | \mathbf{s}_j) \tau(\mathbf{s}_j), \quad (2.49b)$$

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 2.11 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 2.6, 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 2.12, 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* [189], 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.

### 2.6.3 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.

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

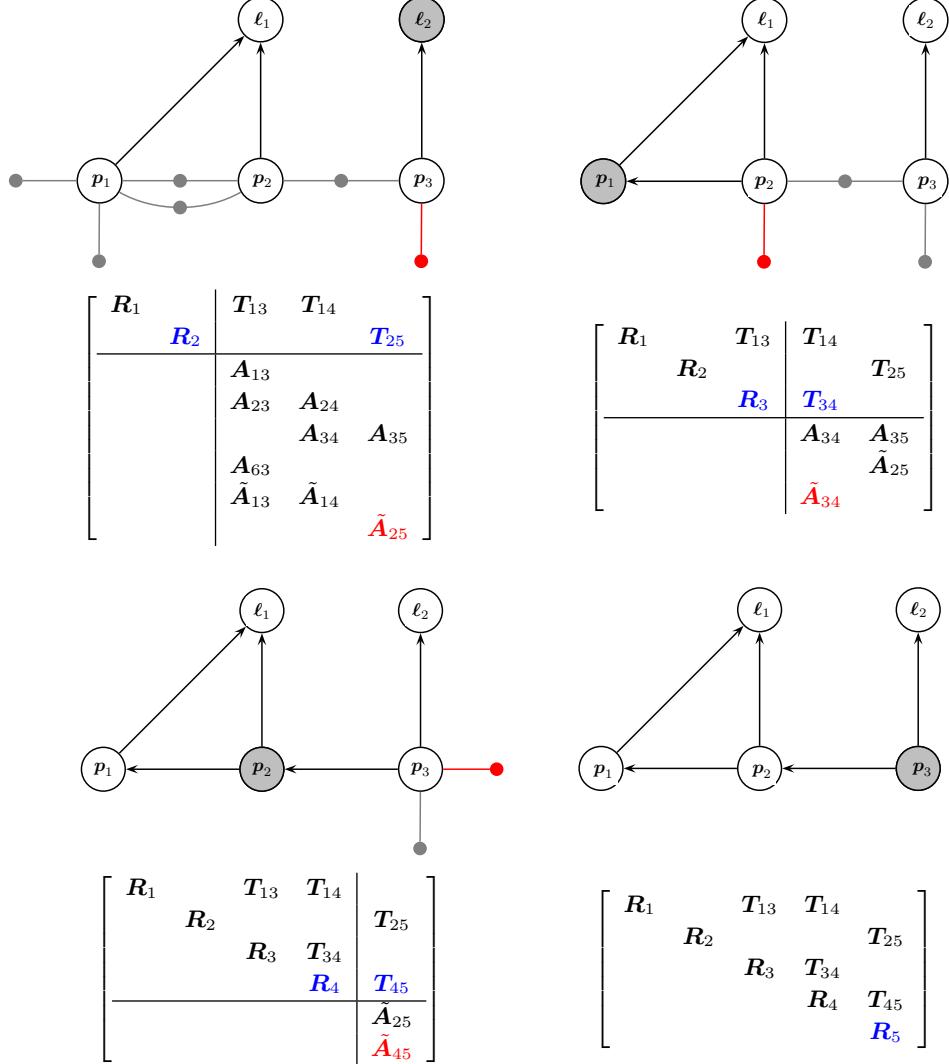


Figure 2.12 The remaining elimination steps for the toy example, completing a full QR factorization. The last step in the bottom right shows the equivalence between the resulting Bayes net and the sparse Cholesky factor  $\mathbf{R}$ .

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:

$$p(\mathbf{x}) = \prod_j p(\mathbf{x}_j | \mathbf{s}_j). \quad (2.50)$$

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

$$p(\mathbf{x}_j | \mathbf{s}_j) = k \exp\left(-\frac{1}{2} \|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2\right), \quad (2.51)$$

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

$$\|\mathbf{R}_j \mathbf{x}_j + \mathbf{T}_j \mathbf{s}_j - \mathbf{d}_j\|_2^2 = (\mathbf{x}_j - \boldsymbol{\mu}_j)^\top \mathbf{R}_j^\top \mathbf{R}_j (\mathbf{x}_j - \boldsymbol{\mu}_j) \triangleq \|\mathbf{x}_j - \boldsymbol{\mu}_j\|_{\Sigma_j}^2, \quad (2.52)$$

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  $\Sigma_j = (\mathbf{R}_j^\top \mathbf{R}_j)^{-1}$ . Hence, the normalization constant  $k = |2\pi \Sigma_j|^{-\frac{1}{2}}$ .

After the elimination step is complete, back-substitution is used to obtain the MAP estimate of each variable. As seen in Figure 2.12, 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,

$$\mathbf{x}_j^* = \mathbf{R}_j^{-1}(\mathbf{d}_j - \mathbf{T}_j \mathbf{s}_j^*), \quad (2.53)$$

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

## 2.7 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 [265]. 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

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

### 2.7.1 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  $\mathbf{x}_j$ ,

$$p(\mathbf{x}) = \prod_j p(\mathbf{x}_j | \boldsymbol{\pi}_j), \quad (2.54)$$

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 2.10, 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:

$$p(\mathbf{x}) = \prod_k p(\mathbf{f}_k | \mathbf{s}_k). \quad (2.55)$$

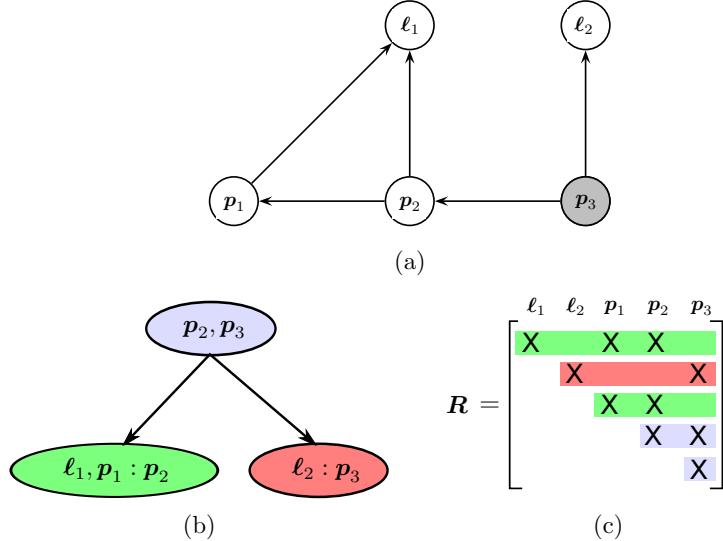


Figure 2.13 The Bayes tree (b) and the associated square root information matrix  $\mathbf{R}$  (c) describing the clique structure in the chordal Bayes net (a) based on our canonical example from Figure 2.2. A Bayes tree is similar to a clique tree, but is better at capturing the formal equivalence 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 2.2) is shown in Figure 2.13. 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.

### 2.7.2 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.

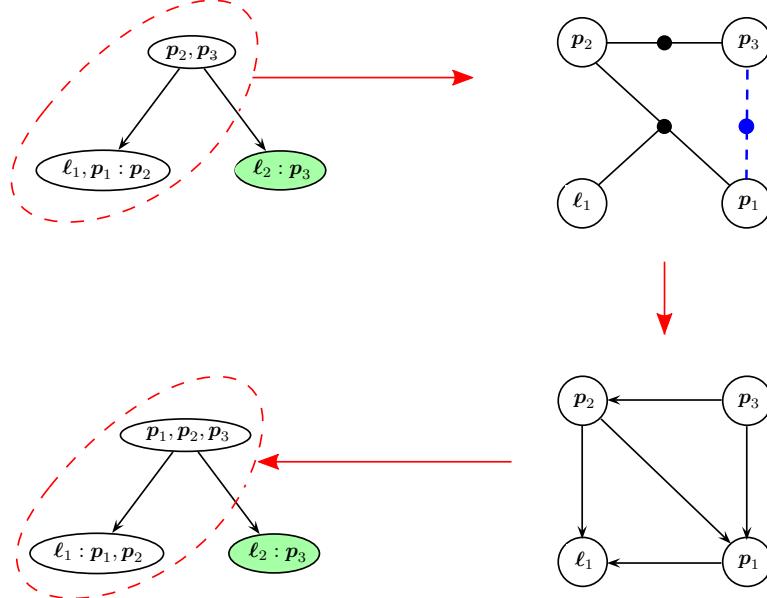


Figure 2.14 Updating a Bayes tree with a new factor, based on the example in Figure 2.13. The affected part of the Bayes tree is highlighted for the case of adding a new factor between  $p_1$  and  $p_3$ . Note that the right branch (green) is not affected by the change. (top right) The factor graph 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 2.14 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 2.15 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. [544]. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated.

### 2.7.3 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[352], used the incremental matrix factorization methods from Golub and Loan [265]. 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 [354]. 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 [155]. 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.

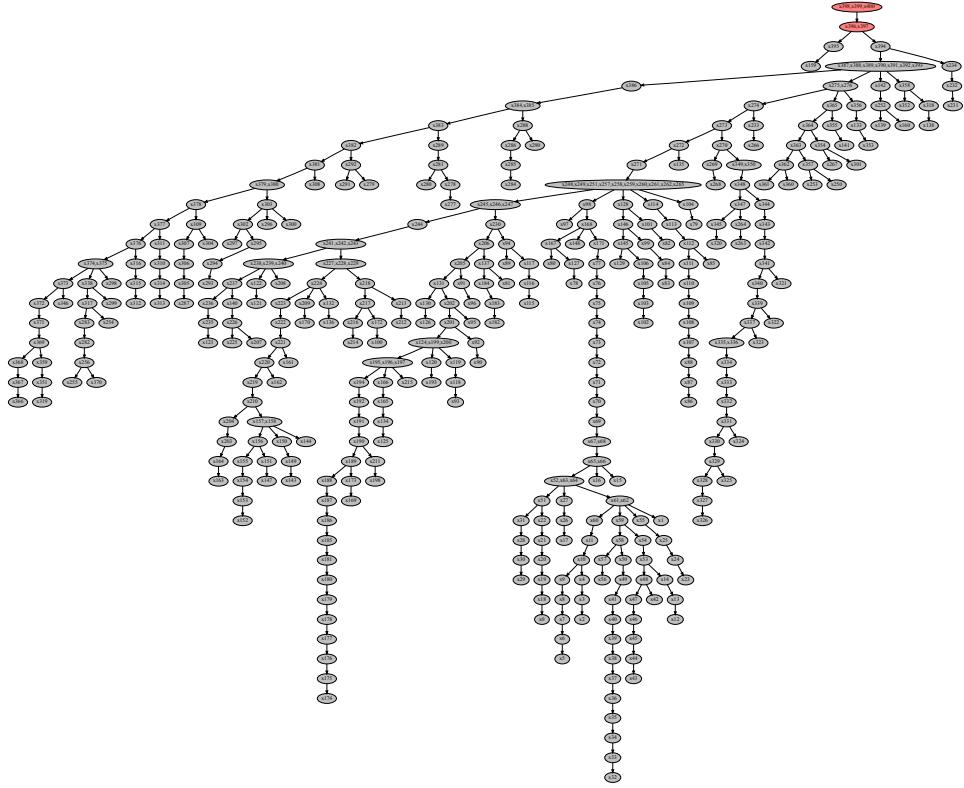


Figure 2.15 An example of the Bayes tree data structure for a small SLAM sequence. The incremental nonlinear least-squares estimation algorithm iSAM2 [354] is based on viewing incremental factorization as editing the graphical model corresponding to the posterior probability of the solution, the Bayes tree. As a robot explores the environment, new measurements often only affect small parts of the tree, and only those parts are re-calculated (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 [354].

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>.

# 3

## Advanced State Variable Representations

Timothy Barfoot, Frank Dellaert, Michael Kaess

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 [35] as the primary reference with some streamlined notation from Sola et al. [658].

### 3.1 Optimization on Manifolds

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 collection of points forming 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 3.1 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.

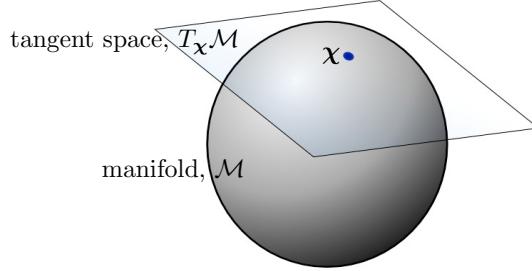


Figure 3.1 For the sphere manifold,  $\mathcal{M}$ , the local tangent plane,  $T_x \mathcal{M}$ , with a local basis provides the notion of local coordinates.

### 3.1.1 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

$$\mathbf{R}_a^b = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad (3.1)$$

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,

$$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}. \quad (3.2)$$

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

using a single matrix multiplication:

$$\underbrace{\begin{bmatrix} \ell^b \\ 1 \end{bmatrix}}_{\tilde{\ell}^b} = \underbrace{\begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathbf{T}_a^b} \underbrace{\begin{bmatrix} \ell^a \\ 1 \end{bmatrix}}_{\tilde{\ell}^a}. \quad (3.3)$$

We refer to  $\tilde{\ell}$  as the *homogeneous* representation of the landmark  $\ell$ .

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*. Thankfully, 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. [164] or Boumal [73] or Barfoot [35]). For additional background on Lie groups in robotics see the seminal work of Chirikjian and Kyatkin [135], Chirikjian [133, 134].

### 3.1.2 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* 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:  $\mathbf{R}_a^c = \mathbf{R}_b^c \mathbf{R}_a^b \in \text{SO}(d)$ .

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. 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

$$\mathbf{R} = \text{Exp}(\theta) = \exp(\theta^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} (\theta^\wedge)^n \in \text{SO}(2), \quad \theta^\wedge = \begin{bmatrix} 0 & -\theta \\ \theta & 0 \end{bmatrix}, \quad \theta \in \mathbb{R}. \quad (3.4)$$

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:  $\theta = \text{Log}(\mathbf{R}) = (\log(\mathbf{R}))^\vee$ .

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:

$$\mathbf{R} = \text{Exp}(\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{\theta}^{\wedge n} \in \text{SO}(3), \quad (3.5a)$$

$$\boldsymbol{\theta}^\wedge = \begin{bmatrix} 0 & -\theta_3 & \theta_2 \\ \theta_3 & 0 & -\theta_1 \\ -\theta_2 & \theta_1 & 0 \end{bmatrix} \in \text{so}(3), \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \in \mathbb{R}^3. \quad (3.5b)$$

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

$$\mathbf{T} = \text{Exp}(\boldsymbol{\xi}) = \exp(\boldsymbol{\xi}^\wedge) \in \text{SE}(d), \quad (3.6a)$$

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\theta}^\wedge & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} \in \text{se}(d), \quad \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^{3(d-1)}, \quad \boldsymbol{\theta} \in \mathbb{R}^{2d-3}, \quad \boldsymbol{\rho} \in \mathbb{R}^d, \quad (3.6b)$$

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.

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 [35].

### 3.1.3 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 2.3.1, 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

$$\mathbf{z}_i = \mathbf{h}_i \left( \mathbf{T}_w^c \tilde{\ell}_i^w \right) + \boldsymbol{\eta}_i, \quad (3.7)$$

where  $\mathbf{T}_w^c \in \text{SE}(3)$  is the pose of the camera with respect to a world frame,  $\tilde{\ell}_i^w$  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

$$\mathbf{T}_w^{c^*} = \arg \min_{\mathbf{T}_w^c} = \sum_i \left\| \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}_w^c \tilde{\ell}_i^w \right) \right\|_{\Sigma_i}^2, \quad (3.8)$$

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>

$$\mathbf{T}_w^c = \mathbf{T}_w^{c^0} \text{Exp}(\boldsymbol{\xi}_w^c). \quad (3.9)$$

Here,  $\boldsymbol{\xi}_w^c \in \mathbb{R}^6$  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

$$\mathbf{T}_w^c = \mathbf{T}_w^{c^0} \oplus \boldsymbol{\xi}_w^c \quad (3.10)$$

implies (3.9). 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

$$\mathbf{T}_w^c \approx \mathbf{T}_w^{c^0} \left( \mathbf{I} + \boldsymbol{\xi}_w^{c^\wedge} \right), \quad (3.11)$$

where we have kept just the terms up to linear in  $\boldsymbol{\xi}_w^c$  from the series form of the matrix exponential. Then, inserting (3.11) into our measurement function (3.7), we have

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \left( \mathbf{I} + \boldsymbol{\xi}_w^{c^\wedge} \right) \tilde{\ell}_i^w \right) + \boldsymbol{\eta}_i. \quad (3.12)$$

This can also be rewritten as

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\ell}_i^w + \mathbf{T}_w^{c^0} \tilde{\ell}_i^w \odot \boldsymbol{\xi}_w^c \right) + \boldsymbol{\eta}_i, \quad (3.13)$$

where  $\odot$  is a (linear) operator for homogeneous points [35]:

$$\tilde{\ell}^\odot = \begin{bmatrix} \ell \\ 1 \end{bmatrix}^\odot = \begin{bmatrix} \mathbf{I} & -\ell^\wedge \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \quad (3.14)$$

We have essentially ‘linearized’ the pose perturbation in (3.13) 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

$$\mathbf{z}_i \approx \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\ell}_i^w \right) + \underbrace{\frac{\partial \mathbf{h}_i}{\partial \ell} \Big|_{\mathbf{T}_w^{c^0} \tilde{\ell}_i^w} \mathbf{T}_w^{c^0} \tilde{\ell}_i^w \odot \boldsymbol{\xi}_w^c}_{\mathbf{H}_i \text{ (chain rule)}} + \boldsymbol{\eta}_i, \quad (3.15)$$

where the chaining of two pieces into the overall Jacobian,  $\mathbf{H}_i$ , is now clear. Looking

<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.

back to (2.21b), we can write the linearized least-squares term (i.e., negative-log factor) for this measurement as

$$\left\| \left( \mathbf{z}_i - \mathbf{h}_i \left( \mathbf{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \right) \right) - \mathbf{H}_i \boldsymbol{\xi}_w^c \right\|_{\Sigma_i}^2, \quad (3.16)$$

where the only unknown is our pose perturbation,  $\boldsymbol{\xi}_w^c$ . After combining this with other factors and then solving for the optimal updates to our state variables, including  $\boldsymbol{\xi}_w^{c^*}$ , we need to update our initial guess,  $\mathbf{T}_w^{c^0}$ . For this, we must return to the perturbation scheme we chose in (3.9) and update according to

$$\mathbf{T}_w^{c^0} \leftarrow \mathbf{T}_w^{c^0} \oplus \boldsymbol{\xi}_w^{c^*}, \quad (3.17)$$

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  $\boldsymbol{\xi}_w^c$ ) is sufficiently small.

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 (3.15) 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). In (3.9), we perturbed our pose variable on the left side, but this was a choice and in some cases perturbing on the right may be preferable.

Stepping back a bit, this approach to optimizing a function of a Lie group member is an example of *Riemannian optimization* [73]. 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 (3.17), 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. [164] or Barfoot et al. [37]).

### 3.1.4 Uncertainty and Lie Groups

We often represent uncertainty in our estimates by considering that the state variables are random, drawn from some distribution. Gaussian distributions are the most common as discussed in Section 2.2.2. For a vector variable,  $\mathbf{x}$ , we can write

$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (3.18)$$

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:

$$\mathbf{R} = \bar{\mathbf{R}} \text{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (3.19)$$

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

$$\mathbf{T} = \bar{\mathbf{T}} \text{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad (3.20)$$

where it is now guaranteed that  $\mathbf{T} \in \text{SE}(d)$  and naturally  $\boldsymbol{\delta}$  and  $\boldsymbol{\Sigma}$  must be appropriately sized. To learn more see, for example, Long et al. [456], Barfoot and Furgale [36].

### **3.1.5 Lie Group Extras**

There is a lot more that we could say about Lie groups [665, 73] 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 5.3.

#### *3.1.5.1 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

$$\mathbf{T} = \mathbf{T}^0 \oplus \boldsymbol{\xi} = \mathbf{T}^0 \text{Exp}(\boldsymbol{\xi}) = \mathbf{T}^0 \exp(\boldsymbol{\xi}^\wedge) \in \text{SE}(d). \quad (3.21)$$

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

$$\boldsymbol{\xi} = \mathbf{T}^0 \ominus \mathbf{T} = \text{Log}(\mathbf{T}^0 \mathbf{T}^{-1}) = \log(\mathbf{T}^0 \mathbf{T}^{-1})^\vee \in \text{se}(d). \quad (3.22)$$

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

#### *3.1.5.2 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

$$(\mathbf{T}^0 \text{Exp}(\boldsymbol{\xi}))^{-1} = \text{Exp}(\boldsymbol{\xi})^{-1} \mathbf{T}^{0^{-1}} = \text{Exp}(-\boldsymbol{\xi}) \mathbf{T}^{0^{-1}}, \quad (3.23)$$

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

### 3.1.5.3 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  $\text{SE}(d)$  transforms a Lie algebra element  $\xi^\wedge \in \text{se}(d)$  to another element of  $\text{se}(d)$  according to a map known as the *inner automorphism* or *conjugation*:

$$\text{Ad}_T \xi^\wedge = T \xi^\wedge T^{-1}. \quad (3.24)$$

We can equivalently express the output of this map as

$$\text{Ad}_T \xi^\wedge = (\text{Ad}(T) \xi)^\wedge, \quad (3.25)$$

where  $\text{Ad}(T)$  linearly transforms  $\xi \in \mathbb{R}^6$  to  $\mathbb{R}^6$ . We will refer to  $\text{Ad}(T)$  as the *adjoint representation* of  $\text{SE}(d)$ .

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

$$\text{Ad}(T) = \text{Ad} \left( \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{R} & \mathbf{t}^\wedge \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}. \quad (3.26)$$

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

$$T \text{Exp}(\xi) = \text{Exp}(\text{Ad}(T)\xi) T, \quad (3.27)$$

which we emphasize does not require approximation.

### 3.1.5.4 Jacobians

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,  $\mathbf{R} \in \text{SO}(d)$ , to angular velocity,  $\boldsymbol{\omega} \in \mathbb{R}^{3(d-1)}$ , is

$$\dot{\mathbf{R}} = \boldsymbol{\omega}^\wedge \mathbf{R}. \quad (3.28)$$

If we parameterize  $\mathbf{R} = \text{Exp}(\boldsymbol{\theta})$ , then we can equivalently write

$$\dot{\boldsymbol{\theta}} = \mathbf{J}(\boldsymbol{\theta}) \boldsymbol{\omega}, \quad (3.29)$$

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

$$\text{Exp}(\boldsymbol{\theta}_1) \text{Exp}(\boldsymbol{\theta}_2) \approx \text{Exp}(\boldsymbol{\theta}_2 + \mathbf{J}(\boldsymbol{\theta}_2)^{-1} \boldsymbol{\theta}_1), \quad (3.30)$$

where  $\theta_1$  is assumed to be ‘small’. The series expression for  $\mathbf{J}(\theta)$  is

$$\mathbf{J}(\theta) = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\theta^\wedge)^n, \quad (3.31)$$

and a closed-form expression can be found in Barfoot [35]. We will overload and write  $\mathbf{J}(\xi)$  for the (left) Jacobian of  $\text{SE}(d)$  where the context should inform which is meant.

### 3.2 Continuous-Time Trajectories

*Continuous-time trajectories* offer a way to represent smooth robot motions. In our development so far, we have assumed 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 be particularly useful when working with high-rate 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 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.

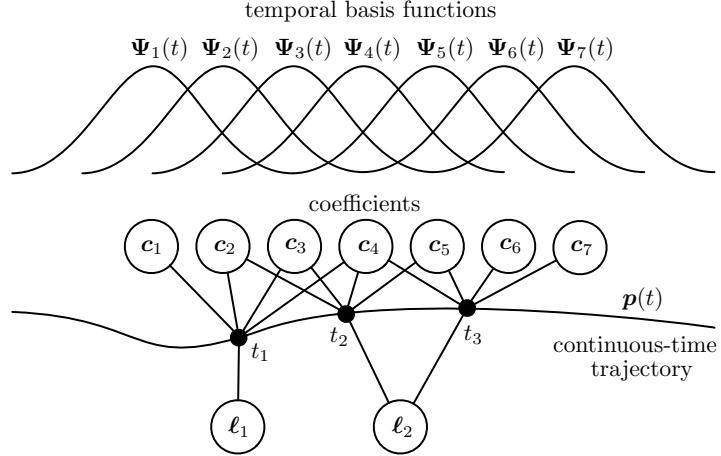


Figure 3.2 A parametric spline can be used to represent a continuous-time trajectory. In this example, the pose at a given time  $p(t)$  is assembled as a weighted sum of known temporal basis functions  $\Psi_k(t)$  with local support; at most four basis functions are nonzero at a given time. This results 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.

### 3.2.1 Splines

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

$$\mathbf{p}(t) = \sum_{k=1}^K \Psi_k(t) \mathbf{c}_k, \quad (3.32)$$

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 3.2. 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

$$\mathbf{z}_i = \mathbf{h}_i(\mathbf{p}(t_i), \ell) + \boldsymbol{\eta}_i. \quad (3.33)$$

Inserting (3.32) we have

$$\mathbf{z}_i = \mathbf{h}_i \left( \sum_{k=1}^K \boldsymbol{\Psi}_k(t_i) \mathbf{c}_k, \boldsymbol{\ell} \right) + \boldsymbol{\eta}_i. \quad (3.34)$$

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  $\mathbf{x}_i = [\mathbf{c}_i^\top \quad \boldsymbol{\ell}^\top]^\top$  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

$$\mathbf{z}_i = \mathbf{h}_i(\mathbf{x}_i) + \boldsymbol{\eta}_i, \quad (3.35)$$

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,

$$\dot{\mathbf{p}}(t) = \sum_{k=1}^K \dot{\boldsymbol{\Psi}}_k(t) \mathbf{c}_k \quad (3.36)$$

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,  $\dot{\boldsymbol{\Psi}}_k(t)$ .

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 (3.32) or (3.36). 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 (3.32) or (3.36) are linear relationships; and, local support in the basis functions implies only the appropriate marginal covariance is needed from the coefficients.

### 3.2.2 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 only pose variables. Using the parametric approach introduced in the previous section, our linearized least-squares term (negative-log factor) will have the form

$$\|(\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i^0)) - \mathbf{H}_i \boldsymbol{\Psi}_i \boldsymbol{\delta}_{c,i}\|_{\Sigma_i}^2, \quad (3.37)$$

where  $\mathbf{x}_i^0$  is the current solution (active coefficients),  $\boldsymbol{\delta}_{c,i}$  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

$$\mathbf{H}_i = \left. \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}} \right|_{\mathbf{x}_i^0}. \quad (3.38)$$

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

$$\boldsymbol{\delta}_c^* = \arg \min_{\boldsymbol{\delta}_c} \left( \|\mathbf{b} - \mathbf{A}\Psi\boldsymbol{\delta}_c\|^2 + \|\boldsymbol{\delta}_c\|^2 \right), \quad (3.39)$$

where we now include a *regularizer term*,  $\|\boldsymbol{\delta}_c\|^2$ , 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 in the last section. The optimal solution will be given by

$$(\Psi^\top \mathbf{A}^\top \mathbf{A} \Psi + \mathbf{I}) \boldsymbol{\delta}_c^* = \Psi^\top \mathbf{A}^\top \mathbf{b}, \quad (3.40)$$

which would allow us to compute the optimal update for the coefficients,  $\boldsymbol{\delta}_c^*$ . 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  $\boldsymbol{\delta}^* = \Psi \boldsymbol{\delta}_c^*$ . With a little bit of algebra, we can show that

$$(\mathbf{A}^\top \mathbf{A} + \mathbf{K}^{-1}) \boldsymbol{\delta}^* = \mathbf{A}^\top \mathbf{b}, \quad (3.41)$$

which is a modified version of the *normal equations*, first introduced in (2.25). The *kernel matrix*,  $\mathbf{K} = \Psi^\top \Psi$ , serves a regularization or smoothing function. The careful reader will notice that (3.41) represents a larger linear system of equations than (3.40) 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 (3.41) 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 (3.41) 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 (3.41), it would seem to

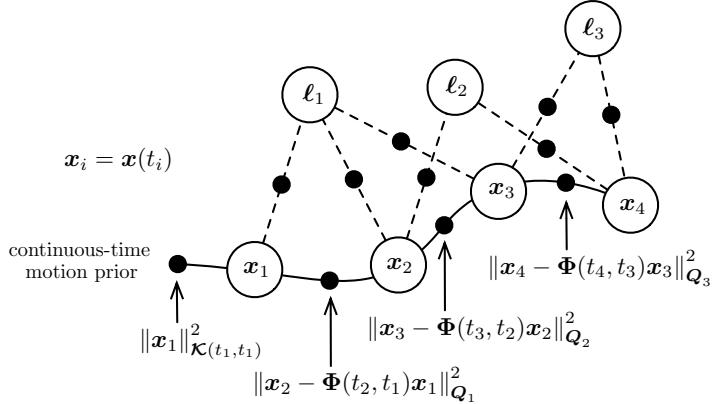


Figure 3.3 Example of Gaussian process (GP) continuous-time factor graph. The motion prior is based on a kernel function derived from a stochastic differential equation (SDE) for Markovian state  $\mathbf{x}(t)$ . This results in a very sparse set of factors: a single unary factor at the initial state and then binary factors linking consecutive states.

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.

### 3.2.3 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 3.3 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>

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{L}\mathbf{w}(t), \quad (3.42)$$

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

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

form:

$$\mathbf{x}(t) = \Phi(t, t_1)\mathbf{x}(t_1) + \int_{t_1}^t \Phi(t, s)\mathbf{L}\mathbf{w}(s) ds, \quad (3.43)$$

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

$$\mathcal{K}(t, t') = \Phi(t, t_1)\mathcal{K}(t_1, t_1)\Phi(t', t_1)^\top + \int_{t_1}^{\min(t, t')} \Phi(t, s)\mathbf{L}\mathbf{Q}\mathbf{L}^\top\Phi(t', s)^\top ds, \quad (3.44)$$

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

$$\mathbf{K} = \Phi\mathbf{Q}\Phi^\top, \quad (3.45)$$

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

$$\Phi = \begin{bmatrix} \mathbf{I} & & & & & \\ \Phi(t_2, t_1) & \mathbf{I} & & & & \\ \Phi(t_3, t_1) & \Phi(t_3, t_2) & \mathbf{I} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ \Phi(t_{M-1}, t_1) & \Phi(t_{M-1}, t_2) & \Phi(t_{M-1}, t_3) & \cdots & \mathbf{I} & \\ \Phi(t_M, t_1) & \Phi(t_M, t_2) & \Phi(t_M, t_3) & \cdots & \Phi(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \quad (3.46)$$

with  $M$  the last measurement time index. However, since it is the *inverse* kernel matrix that we want in (3.41),  $\mathbf{K}^{-1} = \Phi^{-\top}\mathbf{Q}^{-1}\Phi^{-1}$ , 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

$$\Phi^{-1} = \begin{bmatrix} \mathbf{I} & & & & & \\ -\Phi(t_2, t_1) & \mathbf{I} & & & & \\ & -\Phi(t_3, t_2) & \mathbf{I} & & & \\ & & -\Phi(t_4, t_3) & \ddots & & \\ & & & \ddots & \mathbf{I} & \\ & & & & -\Phi(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \quad (3.47)$$

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 (3.41) 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 3.3 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 (3.42), 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

$$\underbrace{\begin{bmatrix} \dot{\mathbf{p}}(t) \\ \dot{\mathbf{v}}(t) \end{bmatrix}}_{\dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{p}(t) \\ \mathbf{v}(t) \end{bmatrix}}_{\mathbf{x}(t)} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}}_{\mathbf{L}} \mathbf{w}(t), \quad (3.48)$$

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* [590]. By making this connection, once we have solved 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.

### 3.2.4 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.

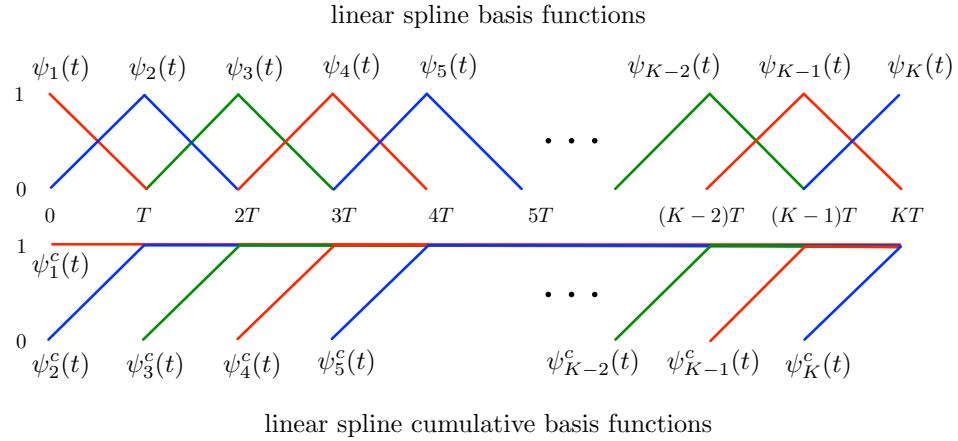


Figure 3.4 Example of linear spline basis functions both in (top) normal and (bottom) cumulative form.

#### 3.2.4.1 Splines on Lie Groups

The key to making splines work on Lie groups is to use a *cumulative formulation*. For a vector space, we can simplify (3.32) by assuming we are using the same basis functions for all degrees of freedom so that we can write

$$\mathbf{p}(t) = \sum_{k=1}^K \psi_k(t) \mathbf{p}_k, \quad (3.49)$$

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

$$\mathbf{p}(t) = \psi_1^c(t) \mathbf{p}_1 + \sum_{k=2}^K \psi_k^c(t) (\mathbf{p}_k - \mathbf{p}_{k-1}), \quad (3.50)$$

where

$$\psi_k^c(t) = \sum_{\ell=k}^K \psi_\ell(t) \quad (3.51)$$

are the *cumulative basis functions*.

For example, if we want to have *linear interpolation* with uniform temporal spac-

ing,  $T$ , the basis functions are

$$\psi_1(t) = \begin{cases} 1 - \alpha_1(t) & 0 \leq t < T \\ 0 & \text{otherwise} \end{cases}, \quad \psi_K(t) = \begin{cases} \alpha_K(t) & (K-1)T \leq t < KT \\ 0 & \text{otherwise} \end{cases}, \quad (3.52)$$

$$k = 2 \dots K-1 : \quad \psi_k(t) = \begin{cases} \alpha_{k-1}(t) & (k-2)T \leq t < (k-1)T \\ 1 - \alpha_k(t) & (k-1)T \leq t < kT \\ 0 & \text{otherwise} \end{cases}, \quad (3.53)$$

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

$$\psi_1^c(t) = 1, \quad \psi_K^c(t) = \begin{cases} 0 & \leq t < (K-1)T \\ \alpha_k(t) & (k-1)T \leq t \end{cases}, \quad (3.54)$$

$$k = 2 \dots K-1 : \quad \psi_k^c(t) = \begin{cases} 0 & t < (k-1)T \\ \alpha_k(t) & (k-1)T \leq t < kT \\ 1 & kT \leq t \end{cases}. \quad (3.55)$$

Figure 3.4 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

$$\mathbf{p}(t) = \mathbf{p}_{k-1} + \psi_k^c(t) (\mathbf{p}_k - \mathbf{p}_{k-1}) \quad (3.56)$$

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

$$\mathbf{T}(t) = \text{Exp}(\psi_k^c(t) \text{Log}(\mathbf{T}_k \mathbf{T}_{k-1}^{-1})) \cdot \mathbf{T}_{k-1}, \quad (3.57)$$

when  $(k-1)T \leq t < kT$ . 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 (3.57) as

$$\mathbf{T}(t) = (\mathbf{T}_k \mathbf{T}_{k-1}^{-1})^{\alpha_k(t)} \mathbf{T}_{k-1}. \quad (3.58)$$

When linearizing expressions involving  $\mathbf{T}(t)$ , we can make use of the optimization

approach introduced in Section 3.1.3. We perturb each of the poses<sup>3</sup> so that

$$\text{Exp}(\boldsymbol{\xi}(t))\mathbf{T}^0(t) = \left(\text{Exp}(\boldsymbol{\xi}_k)\mathbf{T}_k^0\mathbf{T}_{k-1}^{0^{-1}}\text{Exp}(-\boldsymbol{\xi}_{k-1})\right)^{\alpha_k(t)}\text{Exp}(\boldsymbol{\xi}_{k-1})\mathbf{T}_{k-1}^0. \quad (3.59)$$

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 [35], this relationship can be approximated (to first order in the perturbations) as

$$\boldsymbol{\xi}(t) \approx (\mathbf{I} - \mathbf{A}(\alpha_k(t)))\boldsymbol{\xi}_{k-1} + \mathbf{A}(\alpha_k(t))\boldsymbol{\xi}_k, \quad (3.60)$$

where

$$\mathbf{A}(\alpha_k(t)) = \alpha_k(t)\mathbf{J}\left(\alpha_k(t)\mathbf{T}_k^0\mathbf{T}_{k-1}^{0^{-1}}\right)\mathbf{J}\left(\mathbf{T}_k^0\mathbf{T}_{k-1}^{0^{-1}}\right)^{-1} \quad (3.61)$$

and  $\mathbf{J}(\cdot)$  is the left Jacobian of  $\text{SE}(d)$ . We can then use (3.60) 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 (3.15) again, where we rearrange it as an error with slightly simpler notation for the pose and its perturbation as a function of  $t$ :

$$\mathbf{e}_i(t) \approx \mathbf{z}_i - \mathbf{h}_i\left(\mathbf{T}^0(t)\tilde{\ell}_i\right) - \mathbf{H}_i\boldsymbol{\xi}(t). \quad (3.62)$$

It is now a simple matter of substituting (3.60) in for  $\boldsymbol{\xi}(t)$  to produce a linearized error in terms of the bracketing control points:

$$\mathbf{e}_i(t) \approx \mathbf{z}_i - \mathbf{h}_i\left(\mathbf{T}(t)^0\tilde{\ell}_i\right) - \mathbf{H}_i(\mathbf{I} - \mathbf{A}(\alpha_k(t)))\boldsymbol{\xi}_{k-1} - \mathbf{H}_i\mathbf{A}(\alpha_k(t))\boldsymbol{\xi}_k. \quad (3.63)$$

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.

### 3.2.4.2 Gaussian Processes on Lie Groups

To use Gaussian processes on a Lie group, we will again exploit its Lie algebra to do so. Figure 3.5 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-point states [35], similar to splines. Figure 3.6 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

$$\dot{\boldsymbol{\xi}}_k(t) = \mathbf{w}(t), \quad \mathbf{w}(t) = \mathcal{GP}(\mathbf{0}, \mathbf{Q}\delta(t - t')), \quad (3.64)$$

<sup>3</sup> In this case, we are perturbing on the left side instead of the right as shown in Section 3.1.3. The reason is that if the unknown poses represent  $\mathbf{T}_w^s(t)$  (‘sensor’ with respect to ‘world’), we typically apply splines in the ‘sensor’ frame and so choose the perturbations to occur there as well.

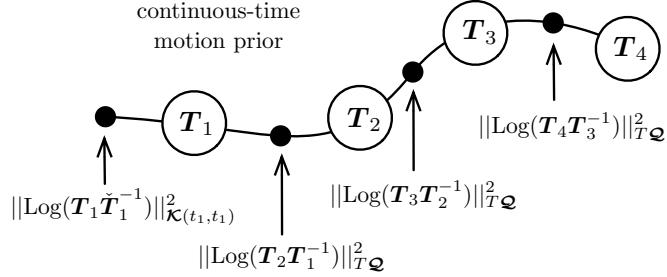


Figure 3.5 Example GP motion prior factors when using a ‘random walk’ model.

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

$$\xi_k(t) = \underbrace{\xi_k(t_k)}_0 + \int_{t_k}^t \mathbf{w}(s) ds \quad (3.65)$$

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

$$\xi_k(t) \sim \mathcal{GP}(\mathbf{0}, \min(t, t') \mathbf{Q}). \quad (3.66)$$

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

$$\Phi^{-1} = \begin{bmatrix} \mathbf{I} & & & \\ -\mathbf{I} & \mathbf{I} & & \\ & \ddots & \ddots & \\ & & -\mathbf{I} & \mathbf{I} \end{bmatrix}, \quad \mathbf{Q} = \text{diag}(\mathcal{K}(t_1, t_1), T\mathbf{Q}, \dots, T\mathbf{Q}). \quad (3.67)$$

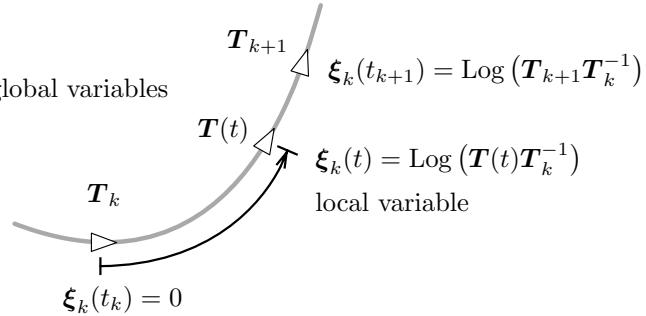


Figure 3.6 When using a GP for continuous-time estimation on Lie groups (e.g.,  $\text{SE}(d)$ ), a local variable,  $\xi_k(t)$ , is defined between control-point states.

The individual errors in terms of the local variables will be

$$\mathbf{e}_k = \begin{cases} \text{Log}(\mathbf{T}_1 \check{\mathbf{T}}_1^{-1}) & k = 1 \\ \boldsymbol{\xi}_{k-1}(t_k) - \boldsymbol{\xi}_{k-1}(t_{k-1}) & k > 1 \end{cases}, \quad (3.68)$$

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

$$\mathbf{e}_k = \begin{cases} \text{Log}(\mathbf{T}_1 \check{\mathbf{T}}_1^{-1}) & k = 1 \\ \text{Log}(\mathbf{T}_k \mathbf{T}_{k-1}^{-1}) & k > 1 \end{cases}. \quad (3.69)$$

Figure 3.5 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 [35]:

$$\mathbf{T}(t) = (\mathbf{T}_k \mathbf{T}_{k-1}^{-1})^{\alpha_k(t)} \mathbf{T}_{k-1}, \quad (3.70)$$

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 (3.69) we can write

$$\begin{aligned} \mathbf{e}_k &= \text{Log}\left(\text{Exp}(\boldsymbol{\xi}_k) \mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}} \text{Exp}(-\boldsymbol{\xi}_{k-1})\right) \\ &\approx \text{Log}\left(\mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}}\right) + \boldsymbol{\xi}_k - \text{Ad}\left(\mathbf{T}_k^0 \mathbf{T}_{k-1}^{0^{-1}}\right) \boldsymbol{\xi}_{k-1}, \end{aligned} \quad (3.71)$$

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 (3.70) 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 (3.63), 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 3.5) to regularize the problem, while the spline approach does not.<sup>4</sup>

<sup>4</sup> Johnson et al. [350] provide a detailed comparison between spline and GP approaches and shows that motion-prior terms can also be introduced to regularize spline methods.

# 4

## Robustness to Incorrect Data Association and Outliers

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

In Chapter 2, 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:

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \|\mathbf{z}_i - \mathbf{h}_i(\mathbf{x}_i)\|_{\Sigma_i}^2, \quad (4.1)$$

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 4.1); these measurements typically induce large perturbations in the estimate  $\mathbf{x}^{\text{MAP}}$  from eq. (4.1). Therefore, we discuss how to reject gross outliers in the SLAM front-end (Section 4.2) and then focus on how to increase robustness to remaining outliers in the SLAM back-end (Section 4.3). We close the chapter with a short review of recent trends and extra pointers to related work (Section 4.4).

### 4.1 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.

#### 4.1.1 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. (4.1) 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 3.

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 2) 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 (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 associations, 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.

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

In the presence of outliers, the estimate resulting from the least-squares formulation (4.1) 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.

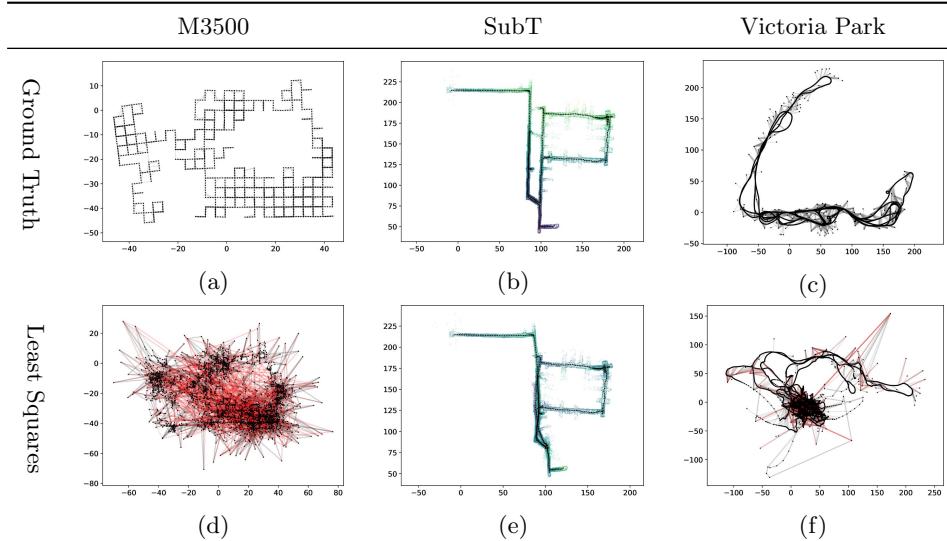


Figure 4.1 SLAM problems with outliers: (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained with the least-squares formulation in the presence of outliers. Inlier measurements are visualized as gray edges, while outliers are visualized as red edges. In the SubT dataset, we also visualize a dense map built from the SLAM pose estimate.

To illustrate this point, Figure 4.1 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 [196]; the dataset includes 682 3D poses and 3278 measurements. The third column is a real-world landmark-SLAM dataset, known as Victoria Park [532]; the dataset includes 7120 2D poses and landmarks, and 17728 measurements. Fig-

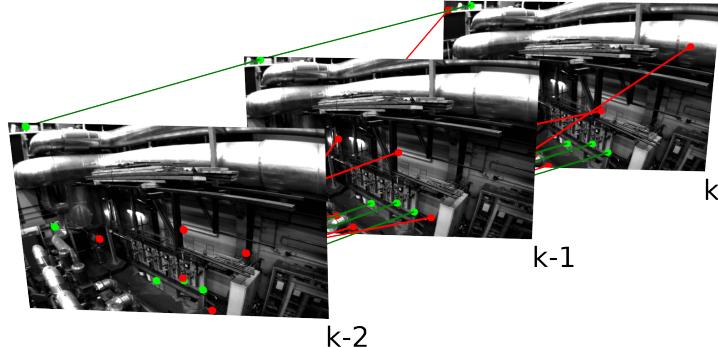


Figure 4.2 Feature tracking across three frames (collected at time  $k - 1$ ,  $k$ , and  $k + 1$ ) in a visual SLAM problem. Inliers are visualized in green, while outliers are visualized in red.

ure 4.1(a)-(c) show the ground truth trajectories for the three problems. Figure 4.1 (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.

## 4.2 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.

### 4.2.1 RANdom SAmple Consensus (RANSAC)

RANSAC is a well-established tool for outlier rejection [220] 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 4.1** (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 4.2). 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 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

$$\mathbf{z}_i(k - 1)^T ([\mathbf{t}_k^{k-1}]^{\times} \mathbf{R}_k^{k-1}) \mathbf{z}_i(k) = 0, \quad (4.2)$$

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

$$C(\mathbf{z}_i, \mathbf{x}) \leq \gamma, \quad (4.3)$$

which states that the correspondences have to satisfy some inequality, which is possibly a function of the unknown state  $\mathbf{x}$ ; in (4.3) 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 (4.2) 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 [290].

constraint (4.3) for some  $\mathbf{x}$ :

$$\begin{aligned} S_{CM}^* = \underset{\mathbf{x}, S \subset M}{\operatorname{argmax}} |S| \\ \text{s.t. } C(\mathbf{z}_i, \mathbf{x}) \leq \gamma, \quad \forall i \in S \end{aligned} \quad (4.4)$$

where  $M$  is the set of initial putative correspondences, and  $|S|$  denotes the cardinality (number of elements) in the subset  $S$  [481]. In words, the optimization (4.4) 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 (4.4) 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 (4.4) is known as *consensus maximization* in computer vision. Note that (4.4) 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 (4.2) only involves the relative pose between two frames rather than the entire SLAM trajectory. At the same time, (4.4) 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 (4.4), it is common to resort to quick heuristics to approximately solve (4.4).

*RANDom SAmple Consensus* (RANSAC) is probably the most well-known approach to find an approximate solution to the consensus maximization problem in (4.4). RANSAC builds on the key assumption that  $\mathbf{x}$  in (4.4) 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 [533]. 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 the 5-point method.

$\gamma$  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 (allowing even thousand 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 context of many SLAM problems, the assumptions of having many inliers and small minimal sets are not always valid.

#### 4.2.2 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 *pair-*

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

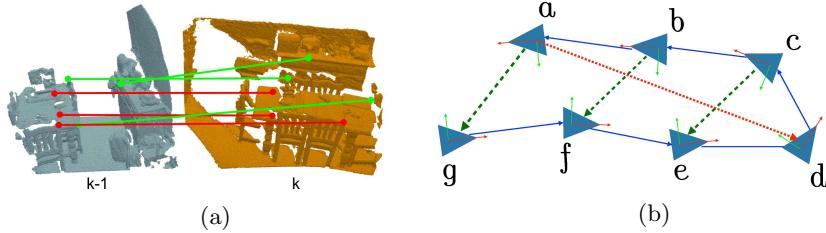


Figure 4.3 (a) 3D-3D correspondences from two RGB-D scans representing two partial views of a scene. The green lines indicate inlier correspondences and the red lines indicate outlier correspondences. (b) Pose graph with outliers in the loop closures. The dashed green lines indicate inlier loop closures while the dotted red line is an outlier loop closure.

*wise 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 [481] and extended beyond pairwise consistency in [649, 224].

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 4.2** (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 4.3(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):

$$\|\mathbf{z}_i(k-1) - \mathbf{z}_j(k-1)\| - \|\mathbf{z}_i(k) - \mathbf{z}_j(k)\| \leq \gamma \quad (4.5)$$

We observe that contrary to the geometric constraints used in RANSAC, the consistency function (4.5) (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 [321].

**Example 4.3** (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 4.3(b)).<sup>8</sup> Therefore, a pair of loop closures  $\mathbf{T}_{ab}$  (between poses  $a$  and  $b$ ) and  $\mathbf{T}_{cd}$  (between poses  $c$  and  $d$ ) must satisfy:

$$\text{dist}(\mathbf{T}_{ab} \cdot \bar{\mathbf{T}}_{bc} \cdot \mathbf{T}_{cd} \cdot \bar{\mathbf{T}}_{da}, \mathbf{I}) \leq \gamma \quad (4.6)$$

where  $\bar{\mathbf{T}}_{bc}$  and  $\bar{\mathbf{T}}_{da}$  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  $\mathbf{T}_{ab} \cdot \bar{\mathbf{T}}_{bc} \cdot \mathbf{T}_{cd} \cdot \bar{\mathbf{T}}_{da}$  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:

$$F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma, \quad (4.7)$$

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

$$\begin{aligned} S_{\text{PCM}}^* &= \underset{S \subset M}{\operatorname{argmax}} |S| \\ \text{s.t. } F(\mathbf{z}_i, \mathbf{z}_j) &\leq \gamma, \quad \forall i, j \in S \end{aligned} \quad (4.8)$$

Problem (4.8) looks for the largest subset  $S$  of measurements such that every pair of measurements in  $S$  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 (4.4): 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  $M$ . Furthermore, the problem admits a graph-theoretic interpretation, which allows solving (4.4) using well-established tools from graph theory, namely, *maximum clique* algorithms.

In order to draw a connection between problem (4.8) 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  $i \in M$  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 [481, 195].

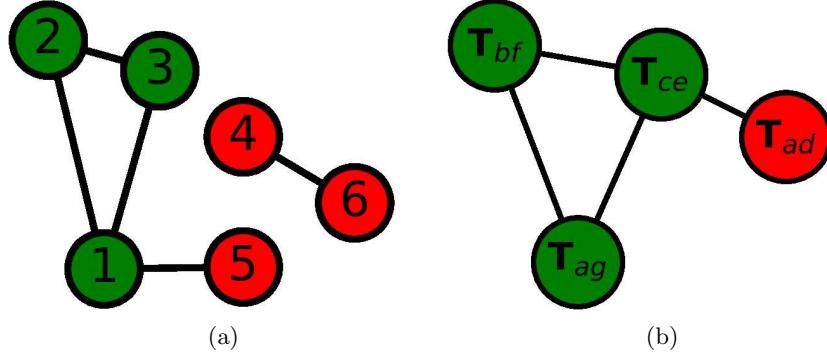


Figure 4.4 (a) Consistency graph of the 3D-3D correspondences example in Figure 4.3(a). (b) Consistency graph of the loop closures for the pose-graph example in Figure 4.3(b)

$F(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma$ . This is typically called the *consistency graph*. Now problem (4.8) asks to select the largest subset of nodes  $S$  such that every pair of nodes in  $S$  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 (4.8) 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 [766] and hard to approximate [847, 217], meaning that finding a solution arbitrarily close to the true solution is also NP-hard. However, dozens of potential solutions 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 [766]. Relatively recent works, *e.g.*, [560], propose maximum clique algorithms that are parallelizable and able to quickly find maximum cliques in 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 broad variety of geometric problems is discussed in [649, 224].

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 4.2), the pairwise consistency function (4.5) 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 [649]).

### 4.3 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 4.1.2, a handful of outliers can lead to completely wrong

<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 [481].

<sup>11</sup> Intuitively, both the geometric constraints (4.3) —even when evaluated at the ground truth  $\mathbf{x}$ — and the pairwise consistency constraints (4.7) 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.

results when using standard least squares estimation. Therefore, it is important to enhance the back-end to be robust to remaining outliers.

In Section 4.1.2, we observed that the use of squared residuals “amplifies” the 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 [330].

M-Estimation (“Maximum-likelihood-type Estimation”) is a framework for robust estimation and suggests replacing the squared loss in eq. (4.1) with a suitably chosen *robust loss* function  $\rho$ :

$$\mathbf{x}^{\text{MAP}} = \arg \min_{\mathbf{x}} \sum_i \mathbf{r}_i(\mathbf{x})^2 \implies \mathbf{x}^{\text{MEST}} = \arg \min_{\mathbf{x}} \sum_i \rho(\mathbf{r}_i(\mathbf{x})). \quad (4.9)$$

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:

$$\frac{\partial f}{\partial \mathbf{x}} = \sum_i \frac{\partial \rho(\mathbf{r}_i)}{\partial \mathbf{r}_i} \cdot \frac{\partial \mathbf{r}_i(\mathbf{x})}{\partial \mathbf{x}} \quad (4.10)$$

From (4.10), 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* [60].

Rather than a single choice of robust loss function, the robust estimation literature provides a “menu” of potential choices. Figure 4.5 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 (4.4).<sup>12</sup> The choice of robust loss is fairly problem-dependent [369, 689]. 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

<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 (4.4).

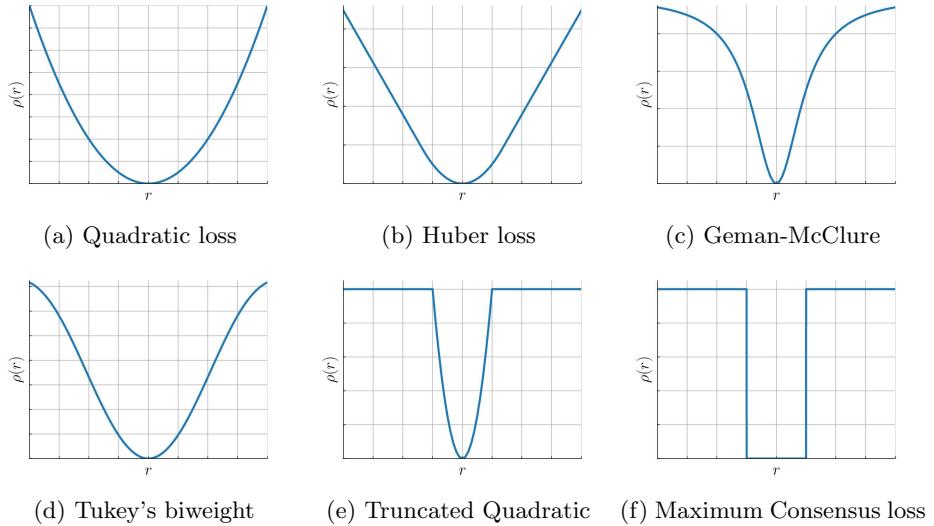


Figure 4.5 Quadratic loss and examples of robust loss functions. The shape of the robust loss functions is controlled by a parameter that controls the separation between inliers and outliers.

considerations. For instance, Huber is often used in bundle adjustment 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 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 4.6(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 4.1. We implemented the gradient descent solver using GTSAM’s NonlinearConjugateGradientOptimizer [162] 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

<sup>13</sup> For instance, RANSAC (Section 4.2.1) 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 4.3.4 below).

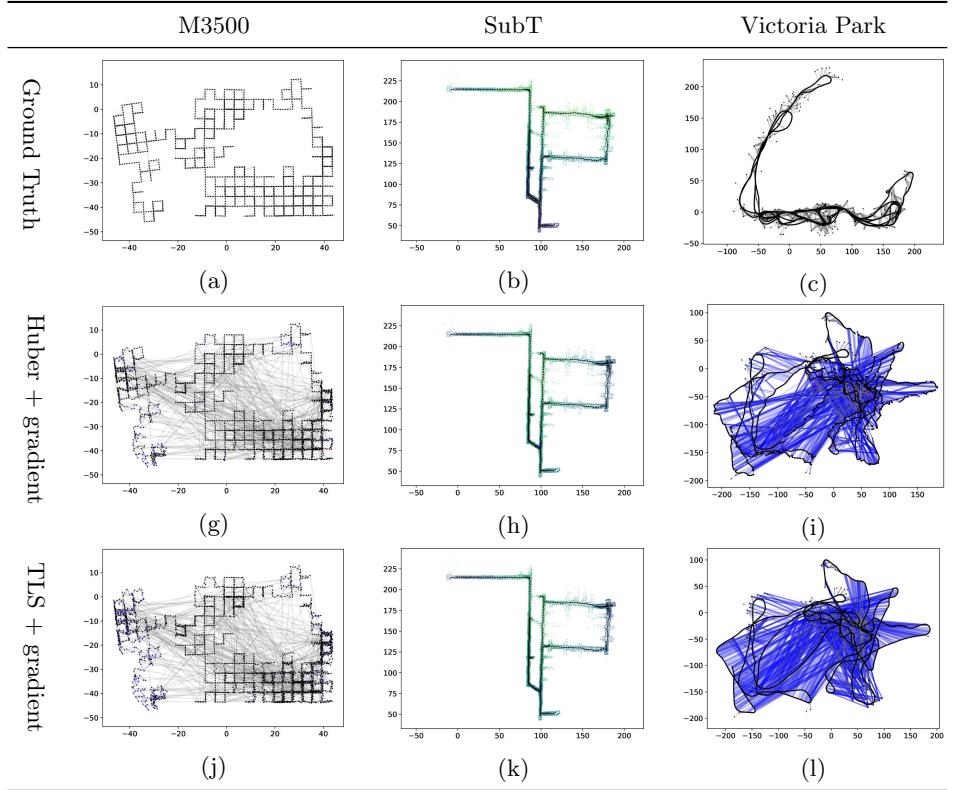


Figure 4.6 Solving SLAM problems with outliers using robust loss functions and gradient descent: (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using gradient descent and Huber loss. (h)-(l) Trajectory estimates obtained using 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”).

(non-robust) least squares optimization (Figure 4.1(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 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 4.6, 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 follow a max-mixture distribution between a Gaussian density (describing the inliers) and a uniform distribution (describing the outliers) [31].

#### 4.3.1 Iteratively Reweighted Least Squares

M-Estimation replaces the least-squares loss by a robust loss  $\rho$  in (4.9) — 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 landspace 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. 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 (4.9) by solving a *weighted* least squares problem at each iteration

$$\mathbf{x}^{(t+1)} = \arg \min_{\mathbf{x}} \sum_i w_i(\mathbf{x}^{(t)}) r_i^2(\mathbf{x}), \quad (4.11)$$

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 (4.9). This implies that the gradient of the robust loss  $\rho$ , shown in (4.10), must match the gradient of the loss in (4.11). By writing down the gradient of (4.11) 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 (4.10), we obtain the IRLS weight update rule

$$w_i(\mathbf{x}^{(t)}) = \frac{1}{2r_i(\mathbf{x}^{(t)})} \frac{\partial \rho(r_i(\mathbf{x}^{(t)}))}{\partial r_i(\mathbf{x}^{(t)})} = \frac{\psi(r_i(\mathbf{x}^{(t)}))}{2r_i(\mathbf{x}^{(t)})}, \quad (4.12)$$

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 (4.11).

Figure 4.7 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 took 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 4.7 and Figure 4.6. The convergence properties of the update rule (4.12) has been studied in [14, 540].

### 4.3.2 Black-Rangarajan Duality

The weight update rule (4.12) is widely used in practice, but its derivation was somewhat heuristic. It also has the issues that (4.12) 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* [60], to solve M-Estimation using IRLS.

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

$$\rho(r_i(\mathbf{x})) := \min\{r_i^2(\mathbf{x}), \beta_i^2\}, \quad (4.13)$$

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 (4.13) can be equivalently written as a sum of two terms by introducing a new weight variable  $w_i \in [0, 1]$

$$\rho(r_i(\mathbf{x})) := \min_{w_i \in [0, 1]} w_i r_i^2(\mathbf{x}) + (1 - w_i) \beta_i^2, \quad (4.14)$$

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 (4.14), the M-Estimation problem (4.9) with a truncated quadratic loss can be reformulated as

$$\min_{\substack{\mathbf{x} \\ w_i \in [0, 1], i=1, \dots, N}} \sum_i [w_i r_i^2(\mathbf{x}) + (1 - w_i) \beta_i^2], \quad (4.15)$$

where we have introduced one  $w_i$  for each measurement residual  $r_i(\mathbf{x})$ . Problem (4.15) 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 (4.15) and hence robustness is ensured.

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

**Theorem 4.4** (Black-Rangarajan Duality [60]). *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*

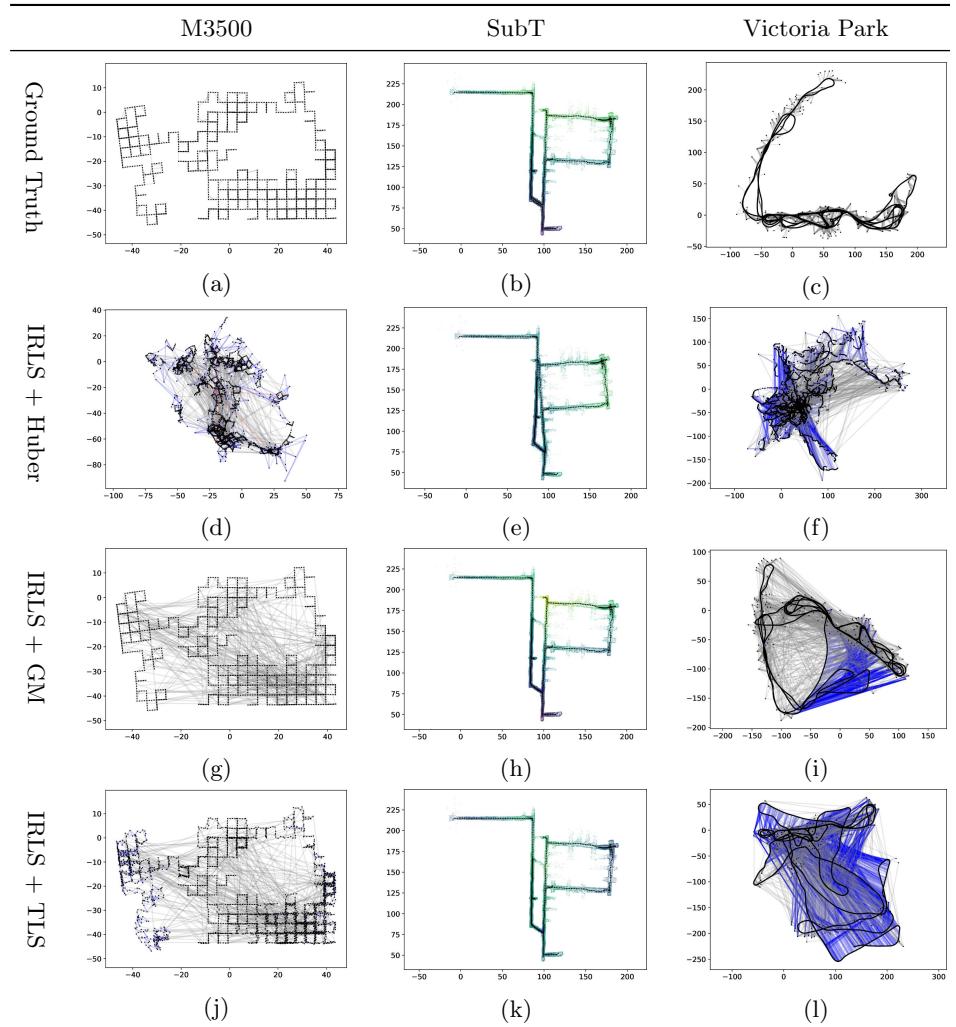


Figure 4.7 Solving SLAM problems with outliers using robust loss functions and Iteratively Re-weighted Least-Squares (IRLS): (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using the Huber loss. (g)-(i) Trajectory estimates obtained 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”).

$\phi''(z) < 0$ , then the M-Estimation problem (4.9) is equivalent to

$$\min_{\mathbf{x}} \sum_{w_i \in [0,1], i=1, \dots, N}^N [w_i r_i^2(\mathbf{x}) + \Phi_\rho(w_i)], \quad (4.16)$$

where  $w_i \in [0, 1]$ ,  $i = 1, \dots, N$  are weight variables associated to each residual  $r_i$ , 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 (4.14) that  $\Phi_\rho(w_i) = (1-w_i)\beta_i^2$ . When  $\rho$  takes other forms, [60] provides a recipe to derive  $\Phi_\rho(w_i)$ . We give an example for the Geman-McClure (G-M) robust loss.

**Example 4.5** (B-R Duality for G-M Loss). Consider the G-M robust loss function

$$\rho(r_i(\mathbf{x})) = \frac{\beta_i^2 r_i^2(\mathbf{x})}{\beta_i^2 + r_i^2(\mathbf{x})}, \quad (4.17)$$

where  $\beta_i^2$  is a noise bound for the  $i$ -th residual similar to (4.13). The outlier process associated to (4.17) is

$$\Phi_\rho(w_i) = \beta_i^2 (\sqrt{w_i} - 1)^2. \quad (4.18)$$

To verify the correctness of (4.18), consider

$$\min_{w_i \in [0,1]} w_i r_i^2(\mathbf{x}) + \Phi_\rho(w_i), \quad (4.19)$$

whose optimal solution is (via setting the gradient of (4.19) to zero)

$$w_i^\star = \left( \frac{\beta_i^2}{r_i^2(\mathbf{x}) + \beta_i^2} \right)^2. \quad (4.20)$$

Plugging (4.20) back to the objective of (4.19) recovers the G-M robust loss (4.17).

### 4.3.3 Alternating Minimization

With the introduction of B-R duality, the IRLS algorithm naturally comes out using a common optimization strategy called *alternating minimization* [714, 56]. 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 (4.16), 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 (4.16) becomes a weighted least squares; analogously, when  $\mathbf{x}$  is fixed, problem (4.16) 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  $w_i$

$$\min_{w_i \in [0,1]} \Phi_\rho(w_i) + w_i r_i^2(\mathbf{x}). \quad (4.21)$$

Problem (4.21) is easy to solve and often admits a closed-form solution. In fact, for the G-M robust loss, the solution of (4.21) is just (4.20). For the truncated quadratic loss, problem (4.21) 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  $w_i^{(t)}$

$$\mathbf{x}^{(t)} \in \arg \min_{\mathbf{x}} \sum_i w_i^{(t)} r_i^2(\mathbf{x}). \quad (4.22)$$

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

$$w_i^{(t+1)} \in \arg \min_{w_i \in [0,1]} \Phi_\rho(w_i) + w_i r_i^2(\mathbf{x}^{(t)}), i = 1, \dots, N. \quad (4.23)$$

The weight update rule obtained via B-R duality actually matches the popular weight update rule (4.12). Interestingly, instantiating the above IRLS algorithm in SLAM using the G-M robust loss leads to the *dynamic covariance scaling* algorithm [15], which has been proposed in the context of outlier-robust SLAM.

#### 4.3.4 Graduated Non-Convexity

The previous section leveraged Black-Rangarajan duality and alternating minimization to derive the IRLS framework that alternates in solving (4.22) and (4.23). 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 (4.9) or how well does  $w_i^{(0)}$  reflect the inlier-outlier membership of each measurement. For example, [650] 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 4.7).

In this section, we introduce the *graduated non-convexity* (GNC) scheme 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 4.6** (GNC Truncated Quadratic Loss). Consider the GNC truncated quadratic loss function

$$\rho_\mu(r_i(\mathbf{x})) = \begin{cases} r_i^2(\mathbf{x}) & \text{if } r_i^2(\mathbf{x}) \in \left[0, \frac{\mu}{\mu+1}\beta_i^2\right] \\ 2\beta_i|r_i(\mathbf{x})|\sqrt{\mu(\mu+1)} - \mu(\beta_i^2 + r_i^2(\mathbf{x})) & \text{if } r_i^2(\mathbf{x}) \in \left[\frac{\mu}{\mu+1}\beta_i^2, \frac{\mu+1}{\mu}\beta_i^2\right] \\ \beta_i^2 & \text{if } r_i^2(\mathbf{x}) \in \left[\frac{\mu+1}{\mu}\beta_i^2, +\infty\right]. \end{cases} \quad (4.24)$$

$\rho_\mu$  is convex for  $\mu$  approaching zero and retrieves the truncated quadratic loss in (4.13) for  $\mu$  approaching infinity.

**Example 4.7** (GNC Geman-McClure Loss). Consider the GNC Geman-McClure loss function

$$\rho_\mu(r_i(\mathbf{x})) = \frac{\mu\beta_i^2 r_i^2(\mathbf{x})}{\mu\beta_i^2 + r_i^2(\mathbf{x})}. \quad (4.25)$$

$\rho_\mu$  is convex for  $\mu$  approaching  $\infty$  and recovers the G-M loss (4.17) when  $\mu = 1$ .

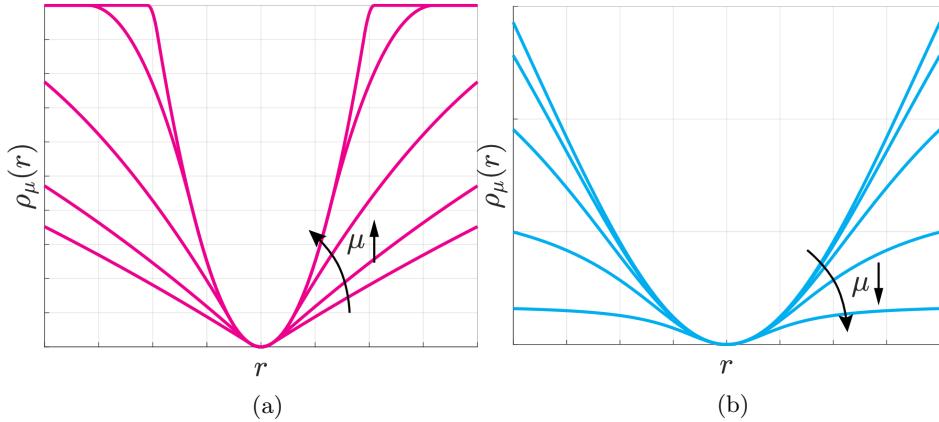


Figure 4.8 Graduated Non-Convexity with control parameter  $\mu$  for (a) Truncated Quadratic loss and (b) Geman-McClure loss.

Figure 4.8(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 (4.24) and (4.25) is that the B-R duality still applies. For the GNC truncated quadratic loss (4.24), 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 (4.25), 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  $w_i^{(t)}$

$$\mathbf{x}^{(t)} \in \arg \min_{\mathbf{x}} \sum_i w_i^{(t)} r_i^2(\mathbf{x}). \quad (4.26)$$

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

$$w_i^{(t+1)} \in \arg \min_{w_i \in [0,1]} \Phi_{\rho_\mu}(w_i) + w_i r_i^2(\mathbf{x}^{(t)}), i = 1, \dots, N. \quad (4.27)$$

- 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 [783]). 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 4.9 showcases the SLAM trajectories obtained by applying GNC on the same three datasets of Figure 4.6, 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 4.6 and Figure 4.7 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 [783, 112]), 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 [649].

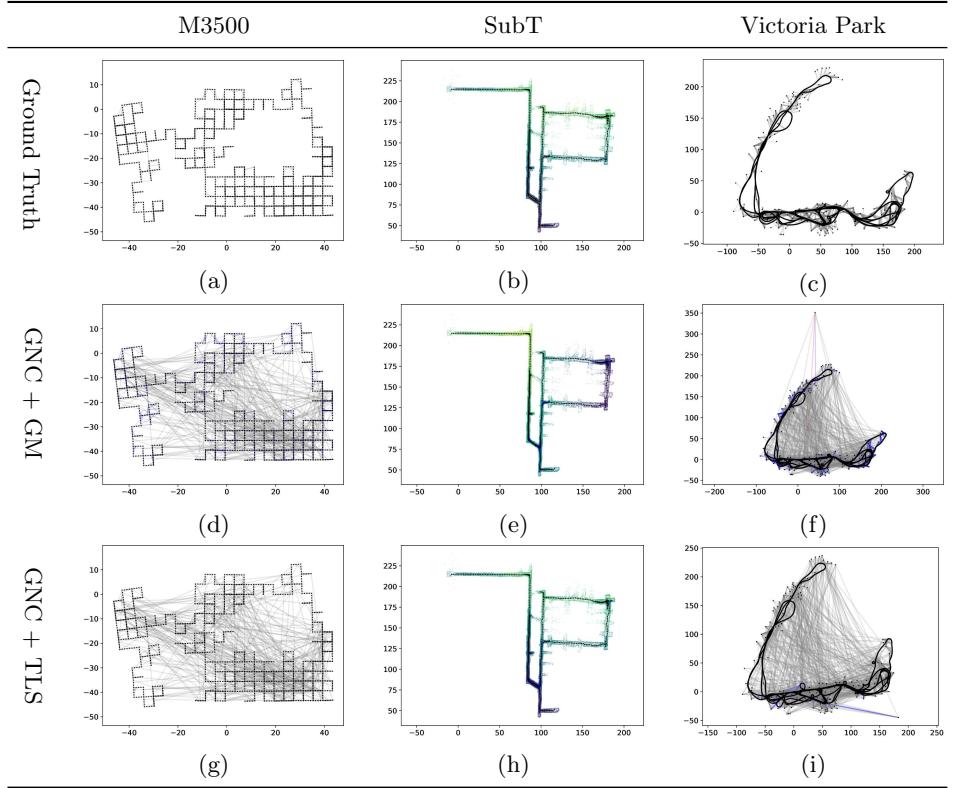


Figure 4.9 Solving SLAM problems with outliers using robust loss functions and graduated non-convexity (GNC): (a)-(c) Ground truth trajectories for the M3500, SubT, and Victoria Park datasets. (d)-(f) Trajectory estimates obtained using the Geman-McClure loss. (g)-(i) Trajectory estimates obtained using 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”).

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 4.1) where the odometry is reliable but there might be 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 instance, 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 [481].

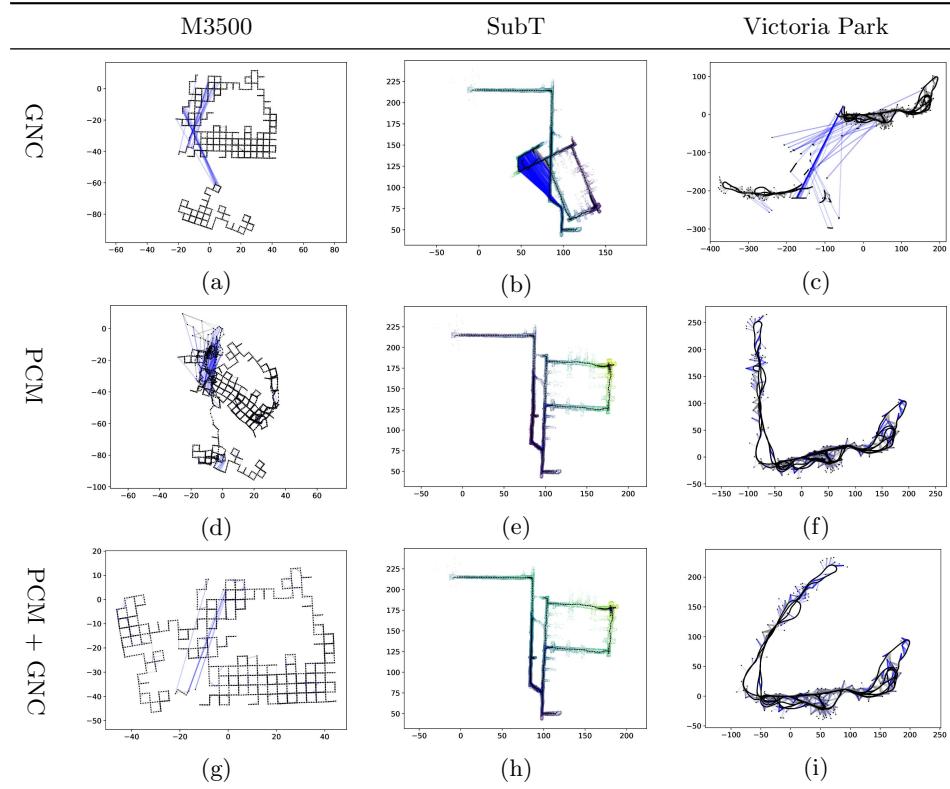


Figure 4.10 SLAM problems with outliers in both the loop closures and landmark measurements, as well as the odometry: (a)-(c) Trajectory estimates obtained using GNC with the truncated quadratic loss. (d)-(f) Trajectory estimates obtained using PCM for front-end outlier rejection followed by least squares 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

<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.

approaches we discussed, in this more complex setting with potentially incorrect odometry measurements, we first modify the pose-graph problems in Figure 4.1 by corrupting two randomly selected odometry measurements. Then we attempt to solve the problem with GNC. In particular, in GNC we fix all odometry measurements except the two potentially corrupted measurements as inliers. Figure 4.10 shows the trajectories obtained by solving the problem with GNC. GNC fails due to the corrupted initialization and converges to a local minima 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 4.10 (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.

#### 4.4 Further References and New Trends

**Consensus Maximization.** While in this chapter we discussed the most basic instantiation of a RANSAC algorithm (according to the initial proposal in [220]), it is worth mentioning that the literature offers many RANSAC variants, including variants that refine estimates through local optimization [142], use better scores (rather than the size of the consensus set) in the RANSAC iterations (*e.g.*, MLESAC [571]), or bias the sampling in the RANSAC iterations (*e.g.*, PROSAC [141]). The recent literature also includes differentiable variants of RANSAC [749] and variants that attempt to find the inliers when the parameter  $\gamma$  in (4.4) is unknown (*e.g.*, [34]). A recent survey and evaluation of RANSAC variants can be found in [712].

Beyond RANSAC, the literature also includes approaches for *exact* consensus maximization, typically based on *branch-and-bound* [49, 291, 832, 433, 659, 131, 338, 87, 788, 787]. 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 4.2.2 was originally proposed in the context of multi-robot SLAM in [481], where this approach showed particular promise. Graph-theoretic outlier rejection has also been investigated in computer vision. Segundo and Artieda [623] build an association graph and find the maximum clique for 2D image feature matching. Perera and Barnes [569] segment objects under rigid body motion with a clique formulation. Leordeanu and Hebert [423] establish image matches by finding strongly-connected clusters in the correspondence graph with an approximate spectral method. Enqvist *et al.* [208] develop an outlier rejection algorithm for

3D-3D and 2D-3D registration based on approximate vertex cover. Yang and Carlone [779, 784] and Parra *et al.* [88] 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.* [412], which perform loop-closure outlier rejection by clustering measurements together and checking for consistency using a Chi-squared-based test. The PCM paper [481], 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 (4.7) involves  $k$  measurements instead of only 2 measurements) in [649, 650, 224]. 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 (4.7) is relaxed to produce continuous weights on the edges of the consistency graph [469, 468]. These methods have been used in practical applications, including subterranean exploration [195], lidar point-cloud localization [470], multi-robot metrics-semantic mapping [699], and global localization in unstructured environments [30].

**Alternating Minimization and Graduated Non-Convexity.** M-estimation has been a popular approach for robust estimation in robotics [67] and vision [629, 115]. Tavish *et al.* [689] investigate the performance of different loss functions. Several papers investigate formulations with auxiliary variables as the one in (4.16), without realizing the connection to M-estimation provided by the Black-Rangarajan duality (Theorem 4.4). For instance, Sünderhauf and Protzel [680, 681] and Agarwal *et al.* [15] augment the problem with latent binary variables responsible for deactivating outliers. Lee *et al.* [421] use expectation maximization. Olson and Agarwal [547] use a max-mixture distribution to approximate multi-modal measurement noise. Recently, Barron [45] proposes a single parametrized function that generalizes a family of robust loss functions in M-estimation. Chebrolu *et al.* [116] design an expectation-maximization algorithm to simultaneously estimate the unknown quantity  $\mathbf{x}$  and choose a suitable robust loss function  $\rho$ . The graduated non-convexity algorithm was first introduced in [61, 60] for outlier rejection in early vision applications; more recently, the algorithm was used for point cloud registration [836, 784], SLAM [783], and other applications [31]. Recently, Peng *et al.* [563] has 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, [563] 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 estimation 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 [780], 3D-3D registration [784], and pose-graph optimization [410, 101]. A fairly general approach to derive certifiable algorithms for problems with outliers is described in [781, 782], while connections with parallel work in statistics is discussed in [100]. 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 [781], 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 7.

# 5

## Differentiable Optimization

Chen Wang, Krishna Murthy Jatavallabhula, and Mustafa Mukadam

### 5.1 Introduction

As presented in Chapter 2, 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 [627, 173, 816] and front-end motion estimation [743, 692]. 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.

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 [691], motion planning [57, 778], SLAM [341, 692], bundle adjustment [686, 816], state estimation [800, 121], and image alignment [473].

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 5.1.1 restates the non-linear least square (NLS) problem. Section 5.2 describes how to differentiate through the NLS problem. Section 5.3 shows how to differentiate problems defined on manifold. Section 5.4 discusses numerical challenges of the above differentiation and introduces related machine learning libraries. Finally, Section 5.5 provides examples of differentiable optimization in contemporary SLAM systems.

### 5.1.1 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 involves a model that is non-linear in the parameters. Beyond our factors graphs in Chapter 2, 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:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}) = \arg \min_{\mathbf{x}} = \frac{1}{2} \sum_i \underbrace{\| w_i \mathbf{c}_i(\mathbf{x}_i) \|_2^2}_{\mathbf{r}_i(\mathbf{x}_i)}, \quad (5.1)$$

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 2, 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 3 that the addition in the update step is more generally a

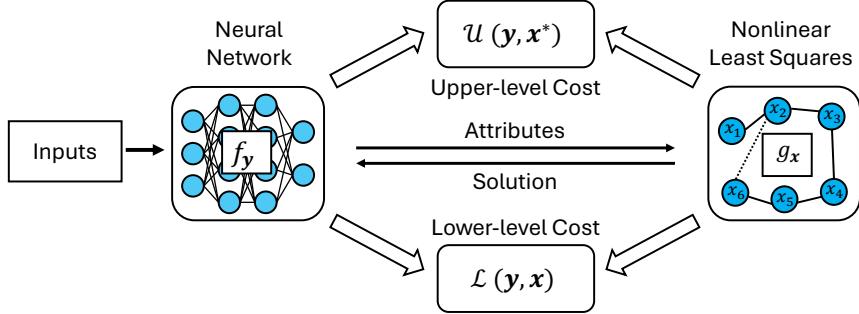


Figure 5.1 A modern SLAM system often involves both neural networks and nonlinear least squares. To eliminate compound errors introduced by optimizing the two modules separately, we can optimize the system in an end-to-end manner by formulating the entire system as a bilevel optimization, which involves an upper-level cost and a lower-level cost.

retraction mapping for variables that belong to a manifold. In the linear system,  $\mathbf{J}_i = [\partial \mathbf{r}_i / \partial \mathbf{x}_i]$  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 2.

## 5.2 Differentiation Through Nonlinear Least Squares

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 5.1. 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:

$$\mathbf{y}^* = \arg \min_{\mathbf{y} \in \Theta} \mathcal{U}(\mathbf{y}, \mathbf{x}^*), \quad (5.2a)$$

$$\text{s. t. } \mathbf{x}^* = \arg \min_{\mathbf{x} \in \Psi} \mathcal{L}(\mathbf{y}, \mathbf{x}), \quad (5.2b)$$

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 5.1.** Imagine a SLAM system that leverages a neural network (parameterized by  $\mathbf{y}$ ) for feature extraction/matching, while utilizing bundle adjustment (BA) for pose estimation (parameterized by  $\mathbf{x}$ ), which take the feature matching as an input. In this example, the UL cost (5.2a) can be feature matching error for optimizing the network, while the LL cost  $L$  (5.2b) 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 (5.2) allows us to further reduce the matching error via back-propagating the BA reprojection errors [816].

**Example 5.2.** Imagine a full SLAM system that uses a neural network for front-end pose estimation, while leverages pose-graph optimization (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 [233].

BLO is a long-standing and well-researched problem [733, 345, 449]. 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 compute the gradient of  $\mathcal{U}$  with respect to the UL variable  $\mathbf{y}$ , which can be written as

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} + \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}, \quad (5.3)$$

where the term  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  involves indirect gradient computation. Since other direct gradient terms in (5.3) are easy to obtain, the challenge of solving a BLO (5.2) 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 [449, 733] and here we list a generic framework incorporating both methods in Algorithm 1. We next explain the unrolled differentiation and implicit differentiation, respectively.

### 5.2.1 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

$$\mathbf{x}_t = \Phi_t(\mathbf{x}_{t-1}; \mathbf{y}), \quad t = 1, \dots, T, \quad (5.4)$$

**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 (5.2b) by a generic optimizer  $\mathcal{O}$  with  $T$  steps.
  - 4:   Efficient estimation of upper-level gradients in (5.3) via  
**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 (5.7).  
**Implicit Differentiation (Algorithm 2):** Compute

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

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).

- 5:   Compute  $\mathbf{y}_{k+1}$  via gradients using  $\hat{\nabla}_{\mathbf{y}_k} \mathcal{U}$ .
  - 6: **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:

$$\Phi_t(\mathbf{x}_{t-1}; \mathbf{y}) = \mathbf{x}_{t-1} - \eta_t \cdot \frac{\partial \mathcal{L}(\mathbf{x}_{t-1}, \mathbf{y})}{\partial \mathbf{x}_{t-1}}, \quad (5.5)$$

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

$$\mathbf{x}^* \approx \mathbf{x}_T = \Phi(\mathbf{y}) = (\Phi_T \circ \dots \circ \Phi_1 \circ \Phi_0)(\mathbf{y}), \quad (5.6)$$

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 (5.2):

$$\min_{\mathbf{y} \in \Theta} \mathcal{U}(\mathbf{y}, \Phi(\mathbf{y})), \quad (5.7)$$

which needs to compute  $\frac{\partial \Phi(\mathbf{y})}{\partial \mathbf{y}}$  via AutoDiff instead of calculating (5.3). 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 [561], and the other corresponds to the forward-mode way [601]. We omit the details of the two approaches of AutoDiff and refer the readers to the AutoDiff libraries such as PyTorch [559] for deep learning and PyPose [732] and Theseus [572] for SLAM. A review of these approaches can also be found in Liu et al. [449].

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

### 5.2.2 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 (5.5) 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. [640] 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 (5.4) to remove the recursive structure [447], *i.e.*,

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{y}} + \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}}, \quad (5.8)$$

where the term  $\frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}}$  is a Hessian that can be calculated from (5.5) as

$$\frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}} = -\frac{\partial^2 \mathcal{L}(\mathbf{x}_0, \mathbf{y})}{\partial \mathbf{x}_0 \partial \mathbf{y}}. \quad (5.9)$$

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 (5.8) as a whole:

$$\frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}_1(\mathbf{y}))}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1(\mathbf{y})}{\partial \mathbf{y}} \approx \frac{\frac{\partial \mathcal{L}(\mathbf{x}_0^+, \mathbf{y})}{\partial \mathbf{y}} - \frac{\partial \mathcal{L}(\mathbf{x}_0^-, \mathbf{y})}{\partial \mathbf{y}}}{2\epsilon}, \quad (5.10)$$

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 [732], which will be introduced in Section 5.4.

### 5.2.3 Implicit Differentiation

It is intuitive that the term  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  in (5.3) is dependent on the LL cost (5.2b), thus *implicit differentiation* can be used to derive a solution to the gradient.

**Example 5.3.** In calculus, *implicit differentiation* refers to the method makes use of the chain rule to differentiate 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

$$\frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}} + \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})} \cdot \frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}} = 0. \quad (5.11)$$

This leads to the indirect gradient  $\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}}$  as

$$\frac{\partial \mathbf{x}^*(\mathbf{y})}{\partial \mathbf{y}} = - \left( \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})} \right)^{-1} \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}}, \quad (5.12)$$

The strength of (5.12) 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 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 5.4.** Assume both UL and LU costs have 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 inversion is more impractical.

Recollect that our goal is to compute the gradient in (5.3), substituting (5.12) into (5.3) gives us:

$$\begin{aligned} \nabla_{\mathbf{y}} \mathcal{U} &= \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \underbrace{\frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{x}^*} \left( \underbrace{\frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{x}^*(\mathbf{y})}}_{(\mathbf{H}^T)^{-1}} \right)^{-1} \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}}}_{\mathbf{v}^T} . \quad (5.13) \\ &= \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \mathbf{q}^T \cdot \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{x}^*(\mathbf{y}) \partial \mathbf{y}} \end{aligned}$$

Then we can solve the linear system  $\mathbf{H}\mathbf{q} = \mathbf{v}$  for  $\mathbf{q}^\top$  by optimizing

$$\mathbf{q}^* = \min \arg_{\mathbf{q}} Q(\mathbf{q}) = \min \arg_{\mathbf{q}} \frac{1}{2} \mathbf{q}^\top \mathbf{H} \mathbf{q} - \mathbf{q}^\top \mathbf{v}, \quad (5.14)$$

using efficient linear solvers such as a simple gradient descent or conjugate gradient method [309]. 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:

$$\mathbf{H}\mathbf{q} = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{x} \partial \mathbf{x}} \cdot \mathbf{q} = \frac{\partial (\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \cdot \mathbf{q})}{\partial \mathbf{x}}, \quad (5.15)$$

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:

$$\mathbf{q}_k = \mathbf{q}_{k-1} - \eta (\mathbf{H}\mathbf{q}_{k-1} - \mathbf{v}), \quad (5.16)$$

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 (5.3) as:

$$\nabla_{\mathbf{y}} \mathcal{U} = \frac{\partial \mathcal{U}(\mathbf{y}, \mathbf{x}^*)}{\partial \mathbf{y}} - \underbrace{\left( \frac{\partial^2 \mathcal{L}(\mathbf{x}^*(\mathbf{y}), \mathbf{y})}{\partial \mathbf{y} \partial \mathbf{x}^*(\mathbf{y})} \cdot \mathbf{q} \right)}_{(\mathbf{H}_{\mathbf{yx}} \cdot \mathbf{q})^\top}^\top, \quad (5.17)$$

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

$$\epsilon \sim \left| \frac{\partial U}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial \mathbf{y}} \right|. \quad (5.18)$$

Nevertheless, it is useful when the implicit gradients contain products of *small* second-order derivatives, which depends on the specific NLS problems.

### 5.3 Differentiation on Manifold

Given that the state of a system within SLAM is bound to evolve on specific manifolds, 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.

#### 5.3.1 Derivatives on the Lie Group

Since we introduced the basic concepts of Lie group, Lie algebra, and their basic operations (*e.g.*, exponential and logarithmic maps) in Chapter 3, 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:

$$\chi = \exp(\tau^\wedge) \Leftrightarrow \tau^\wedge = \log(\chi), \quad (5.19)$$

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  $\chi$ :

$$\chi = \text{Exp}(\tau) \Leftrightarrow \tau = \text{Log}(\chi), \quad (5.20)$$

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 capture the concept of displacement on the manifold, as described in (5.21) and (5.22) below:

$$\begin{aligned} \chi_2 &= \chi_1 \oplus \tau \triangleq \chi_1 \circ \text{Exp}(\tau), \\ \tau &= \chi_2 \ominus \chi_1 \triangleq \text{Log}(\chi_1^{-1} \circ \chi_2). \end{aligned} \quad (5.21)$$

The placement of  $\tau$  on the right-hand side in (5.21) signifies that it is expressed in the local frame at  $\chi_1$ . Conversely, the left operators in (5.22) reflect a global frame perspective:

$$\begin{aligned} \chi_2 &= \varepsilon \oplus \chi_1 \triangleq \text{Exp}(\varepsilon) \circ \chi_1, \\ \varepsilon &= \chi_2 \ominus \chi_1 \triangleq \text{Log}(\chi_2 \circ \chi_1^{-1}), \end{aligned} \quad (5.22)$$

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  $\mathfrak{m}$ :

$$\begin{aligned}\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} &\triangleq \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{f(\boldsymbol{\chi} \oplus \boldsymbol{\tau}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\tau}} \\ &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{f(\boldsymbol{\chi} \circ \text{Exp}(\boldsymbol{\tau})) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\tau}} \\ &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{\text{Log}(f(\boldsymbol{\chi})^{-1} \circ f(\boldsymbol{\chi} \circ \text{Exp}(\boldsymbol{\tau})))}{\boldsymbol{\tau}}.\end{aligned}\quad (5.23)$$

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  $\boldsymbol{\tau} = 0$ :

$$\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} = \mathbf{J}_R = \left. \frac{\partial g(\boldsymbol{\tau})}{\partial \boldsymbol{\tau}} \right|_{\boldsymbol{\tau}=0}. \quad (5.24)$$

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:

$$\begin{aligned}\frac{\partial f(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}} &\triangleq \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{f(\boldsymbol{\epsilon} \oplus \boldsymbol{\chi}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\epsilon}} \\ &= \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \ominus f(\boldsymbol{\chi})}{\boldsymbol{\epsilon}} \\ &= \lim_{\boldsymbol{\epsilon} \rightarrow 0} \frac{\text{Log}(f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \circ f(\boldsymbol{\chi})^{-1})}{\boldsymbol{\epsilon}} \\ &= \left. \frac{\partial \text{Log}(f(\text{Exp}(\boldsymbol{\epsilon}) \circ \boldsymbol{\chi}) \circ f(\boldsymbol{\chi})^{-1})}{\partial \boldsymbol{\epsilon}} \right|_{\boldsymbol{\epsilon}=0}.\end{aligned}\quad (5.25)$$

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:

$$\boldsymbol{\Sigma}_{\boldsymbol{\chi}} \triangleq \mathbb{E}[\boldsymbol{\tau}\boldsymbol{\tau}^T] = \mathbb{E}[(\boldsymbol{\chi} \ominus \boldsymbol{\chi}_1)(\boldsymbol{\chi} \ominus \boldsymbol{\chi}_1)^T]. \quad (5.26)$$

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 5.5.** 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  $SO(3)$ :

$$\hat{\mathbf{R}} = \arg \min_{\mathbf{R} \in SO(3)} f(\mathbf{R}, \mathbf{R}_{\text{IMU}}, \mathbf{R}_{\text{Cam}}), \quad (5.27)$$

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 detailed as:

$$f(\mathbf{R}) = \|\text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R})\|^2 + \|\text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R})\|^2. \quad (5.28)$$

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:

$$\begin{aligned} \nabla f(\mathbf{R}) &= 2 \left( \frac{\partial \text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R})}{\partial \mathbf{R}} \right)^\top \text{Log}(\mathbf{R}_{\text{IMU}}^{-1} \mathbf{R}) \\ &\quad + 2 \left( \frac{\partial \text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R})}{\partial \mathbf{R}} \right)^\top \text{Log}(\mathbf{R}_{\text{Cam}}^{-1} \mathbf{R}). \end{aligned} \quad (5.29)$$

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:

$$\mathbf{R}_{k+1} = \mathbf{R}_k \text{Exp}(-\alpha \nabla f(\mathbf{R})), \quad (5.30)$$

where  $\alpha$  is the step size. This iterative process continues until the cost function  $f(\mathbf{R})$  converges to a minimum, providing an optimal pose estimate  $\hat{\mathbf{R}}$  that aligns with the sensor measurements.

### 5.3.2 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:

$$\frac{\partial \mathcal{Z}}{\partial \chi} = \frac{\partial \mathcal{Z}}{\partial \mathcal{Y}} \frac{\partial \mathcal{Y}}{\partial \chi}, \quad (5.31)$$

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 (5.23) for the right Jacobian  $\mathbf{J}_R$ , which leads to:

$$\begin{aligned} \frac{\partial \chi^{-1}}{\partial \chi} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi^{-1})^{-1}(\chi \text{Exp}(\tau))^{-1})}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\chi \text{Exp}(\tau)^{-1} \chi^{-1})}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{(\chi(-\tau)^\wedge \chi^{-1})^\vee}{\tau}. \end{aligned} \quad (5.32)$$

**Jacobians of composition** can be derived through the application of the function  $f(\chi) = \chi \circ \chi_1$  with the Equation (5.23). The derivative of the composition operator  $\chi \circ \chi_1$  with respect to  $\chi$  is:

$$\begin{aligned} \frac{\partial(\chi \circ \chi_1)}{\partial \chi} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi \chi_1)^{-1}(\chi \text{Exp}(\tau) \chi_1))}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\chi_1^{-1} \text{Exp}(\tau) \chi_1)}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{(\chi_1^{-1} \tau^\wedge \chi_1)^\vee}{\tau}. \end{aligned} \quad (5.33)$$

The derivative of the composition operator  $\chi \circ \chi_1$  with respect to  $\chi_1$  is:

$$\begin{aligned} \frac{\partial(\chi \circ \chi_1)}{\partial \chi_1} &\triangleq \lim_{\tau \rightarrow 0} \frac{\text{Log}((\chi \chi_1)^{-1}(\chi \chi_1 \text{Exp}(\tau)))}{\tau} \\ &= \lim_{\tau \rightarrow 0} \frac{\text{Log}(\text{Exp}(\tau))}{\tau} \\ &= \mathbf{I}. \end{aligned} \quad (5.34)$$

**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:

$$\mathbf{J}_r(\tau) \triangleq \frac{\tau \partial \text{Exp}(\tau)}{\partial \tau}. \quad (5.35)$$

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:

$$\mathbf{J}_l(\tau) \triangleq \frac{\epsilon \partial \text{Exp}(\tau)}{\partial \tau}. \quad (5.36)$$

**Jacobians of group action** depends on the specific group action set  $v \in \mathcal{V}$ . The group action is defined as:

$$\begin{aligned}\mathbf{J}_{\chi}^{\chi \cdot v} &\triangleq \frac{\chi D\chi \cdot v}{D\chi}, \\ \mathbf{J}_v^{\chi \cdot v} &\triangleq \frac{v D\chi \cdot v}{Dv},\end{aligned}\tag{5.37}$$

where  $\chi \in \mathcal{M}$  and  $v \in \mathcal{V}$ .

**Example 5.6.** Consider a robotic arm with two joints,  $\mathbf{R}_1$  and  $\mathbf{R}_2$ , each represented by an element in  $SO(3)$ . The final orientation of the robot's end-effector is determined by the composition of the joint rotations:

$$\mathbf{R} = \mathbf{R}_1 \circ \mathbf{R}_2.\tag{5.38}$$

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:

$$\begin{aligned}\frac{\partial(\mathbf{R}_1 \circ \mathbf{R}_2)}{\partial \mathbf{R}_1} &= \lim_{\boldsymbol{\tau} \rightarrow 0} \frac{(\mathbf{R}_2^{-1} \boldsymbol{\tau}^\wedge \mathbf{R}_2)^\vee}{\boldsymbol{\tau}}, \\ \frac{\partial(\mathbf{R}_1 \circ \mathbf{R}_2)}{\partial \mathbf{R}_2} &= \mathbf{I}.\end{aligned}\tag{5.39}$$

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  $\mathbf{R}_1$ .

## 5.4 Modern Libraries

### 5.4.1 Numerical Challenges of Automatic Differentiation

Automatic Differentiation (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 delves into the specific numerical issues that arise when using automatic differentiation for manifold-based optimization tasks. Particular attention will be 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. For simplicity, we will take the PyPose library [732] as an example, which defines a general

data structure, `LieTensor` for Lie Group and Lie Algebra. Specifically, we will show its numerical challenges and how PyPose tackle this challenge.

**Analytical Foundations of Exponential Mapping to 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:

$$\text{Exp}(\boldsymbol{\nu}) = \left[ \sin\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \frac{\boldsymbol{\nu}^\top}{\|\boldsymbol{\nu}\|}, \cos\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \right]^\top \quad (5.40)$$

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 (5.40) for the Exp and Log mapping [693]. 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.

$$\text{Exp}(\boldsymbol{\nu}) = \begin{cases} \left[ \boldsymbol{\nu}^T \gamma_e, \cos\left(\frac{\|\boldsymbol{\nu}\|}{2}\right) \right]^\top & \text{if } \|\boldsymbol{\nu}\| > \text{eps} \\ \left[ \boldsymbol{\nu}^T \gamma_o, 1 - \frac{\|\boldsymbol{\nu}\|^2}{8} + \frac{\|\boldsymbol{\nu}\|^4}{384} \right]^\top & \text{otherwise,} \end{cases} \quad (5.41)$$

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 in rotation representation, such as in 3D graphics, robotics, and aerospace engineering.

`LieTensor` is different from the existing libraries in several aspects: (1) PyPose supports auto-diff 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 [693] 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 [801], and Theseus only support Lie groups, while Py-

Pose 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 [2] and the tutorial of PyPose is available in [4]. The usages of a `LieTensor` and its automatic differentiation can be found at <https://github.com/pypose/slambook-snippets/blob/main/lietensor.ipynb>.

#### 5.4.2 Implementation of Differentiable Optimization

To enable end-to-end learning with bilevel optimization, one need to integrate general optimizers beyond the gradient-based methods such as SGD [602] and Adam [388] required by neural methods, since many problems in SLAM such as bundle adjustment and factor graph optimization require other optimizations algorithms such as constrained or 2<sup>nd</sup>-order optimization [39]. Moreover, practical problems have outliers, hence one needs to robustify the loss as described in Chapter 4. Next we consider an Iteratively Reweighted Least Squares (IRLS) approach to SLAM as introduced in Section 4.3, 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:

$$\min_{\mathbf{y}} \sum_i (\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i)^T \boldsymbol{\Sigma}_i (\mathbf{h}_i(\mathbf{x}_i) - \mathbf{z}_i), \quad (5.42)$$

where  $\mathbf{h}(\cdot)$  is a regression model (`Module`),  $\mathbf{x} \in \mathbb{R}^n$  is the parameters to be optimized,  $\mathbf{h}_i$  deontes prediction for the  $i$ -th input sample,  $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$  is a square information matrix. The solution to (5.42) 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:

$$\sum_i (\boldsymbol{\Lambda}_i + \lambda \cdot \text{diag}(\boldsymbol{\Lambda}_i)) \boldsymbol{\delta}_t = - \sum_i \mathbf{J}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i, \quad (5.43)$$

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`:

$$\mathbf{A} \cdot \boldsymbol{\delta}_t = \boldsymbol{\beta}, \quad (5.44)$$

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 [109] or preconditioned conjugate gradient (PCG) [309] solver. In practice, one often introduces robust `kernel` functions  $\rho : \mathbb{R} \mapsto \mathbb{R}$  into (5.42) to

reduce the effect of outliers:

$$\min_{\mathbf{y}} \sum_i \rho(\mathbf{r}_i^T \boldsymbol{\Sigma}_i \mathbf{r}_i), \quad (5.45)$$

where  $\rho$  is designed to down-weigh measurements with large residuals  $\mathbf{r}_i$ . In this case, we need to adjust (5.43) to account for the presence of the robust kernel. A popular way is to use an IRLS method, Triggs' correction [710], which is also adopted by the Ceres [19] 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 be unstable [710]. 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:

$$\mathbf{r}_i^\rho = \sqrt{\rho'(c_i)} \mathbf{r}_i, \quad \mathbf{J}_i^\rho = \sqrt{\rho'(c_i)} \mathbf{J}_i, \quad (5.46)$$

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 [1], while IRLS was introduced in Section 4.3.

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 [459] 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 [3]. The usages of a 2nd-order optimization can be found at <https://github.com/pypose/slambook-snippets/blob/main/optimization.ipynb>.

### 5.4.3 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 [543], 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 [276], a high performance C++ linear algebra library, has been used in many projects such as TensorFlow [7], Ceres [19], GTSAM [161], and g<sup>2</sup>o [271]. ArrayFire [480], 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 [148], OpenNN [549], and MATLAB [487], provide primitive tools for researchers to develop neural networks. However, they only support CPU computation and lack concise APIs, which plague engineers using them in applications. A few years later, deep learning frameworks such as Chainer [704], Theano [22], and Caffe [346] 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 [7], PyTorch [559], and MXNet [123], 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 [286] introduced differentiable transformation layers into Torch-based framework, leading to end-to-end geometric learning. JAX [75] 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 [573] and CvxpypLayer [20] showed how to embed differentiable optimization within deep neural networks. PyPose [732] 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 [19] is an open-source C++ library for large-scale nonlinear least squares optimization problems and has been widely used in SLAM. Pyomo [289] and JuMP [191] 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 [29] 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 [271] and GTSAM [161] 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 [731] 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 [392] 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 [262], which provides standard interfaces for different optimal control solvers and can be

extended to a broad class of dynamical systems in robotic applications. Drake [690] 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.

### 5.5 Final Considerations & Recent Trends

Deep learning methods have witnessed significant development in recent years [828]. As data-driven approaches, they are believed to perform better on visual tracking than traditional handcrafted features. Most studies on the subject employed end-to-end structures, including both supervised methods such DeepVO [741] and TartanVO [743] and unsupervised methods such as UnDeepVO [437] and Unsupervised VIO [748]. 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-consuming process [742].

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 Bundle Adjustment (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 [685] and DROID-SLAM [692]. Additionally, some works focused on compressing image features into codes (embedded features) and optimizing the pose-code graph during inference such as DeepFactors [153]. Furthermore, DiffPoseNet [554] 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” bundle adjustment, thus the supervision is from the ground truth poses. In this case, BA is just a special layer of the network. Recently, iSLAM [233] 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 [660], local planning [778], global planning [124], feature matching [816], and multi-robot routing [282].

# 6

## 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 is known, and the objective is to construct a dense map of its surroundings. Indeed, typical approaches first solve for the robot trajectory using the SLAM backend —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 2 that 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. The chapter continues with an introduction to fundamental representation elements and data structures in Section 6.2, that we then tailor to specific applications in Section 6.4. 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 6.5.

### 6.1 Range Sensing Preliminaries

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

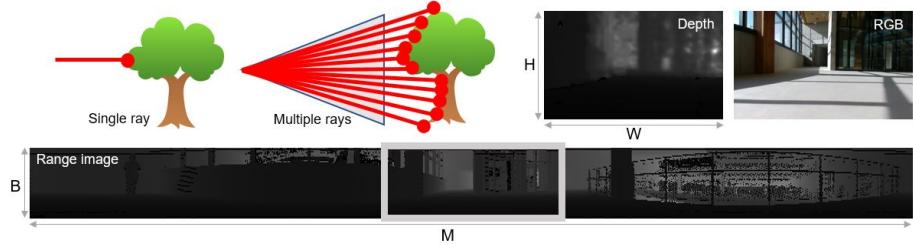


Figure 6.1 Single ray and multi-ray types for LiDAR sensors. Sample data from real-world is visualized to show RGB, depth, and LiDAR range image.

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.

### 6.1.1 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 [605]. 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:

$$r = \frac{c(t_{\text{detect}} - t_{\text{emit}})}{2}. \quad (6.1)$$

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.

Another commonly used range sensor 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 10

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 TOF less affected by interferences, supporting outdoor applications.

### 6.1.2 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.

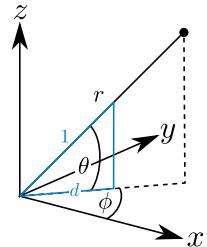
For conversion from LiDAR, the sensors provide an intrinsic calibration for each beam  $(\phi_{i,j}, \theta_{i,j})$ , where  $1 \leq i < B$  and  $1 \leq j < M$ , consisting of the azimuthal angle  $\phi_{i,j} \in [0, 2\pi]$  and polar/inclination angle  $\theta_{i,j} \in [-\pi, \pi]$  as depicted in Figure 6.2. 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:

$$x = r_{i,j} \cos(\theta_{i,j}) \cos(\phi_{i,j}) \quad (6.2)$$

$$y = r_{i,j} \cos(\theta_{i,j}) \sin(\phi_{i,j}) \quad (6.3)$$

$$z = r_{i,j} \sin(\theta_{i,j}) \quad (6.4)$$

Figure 6.2



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:

$$\mathbf{p} = \mathbf{K}^{-1} \mathbf{x}. \quad (6.5)$$

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 longer

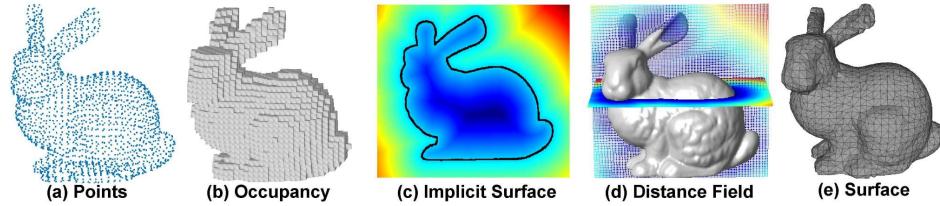


Figure 6.3 Examples of common dense representations.

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.*, inertial measurement unit (IMU) measurements or odometry information, together with the information of the per-beam time is necessary to undistort an LiDAR point cloud to account for the motion of the sensor [728, 165, 791]. For more details on motion distortion and compensation, refer to Chapter 10.

## 6.2 Foundations of Mapping

A map, generated with sensors' information and data processing approaches, is a symbolic structure that models the environment [698, 90]. 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 6.3). 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 [517, 582] 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 [337, 542, 597]. Similarly, path planning tasks require dense information such as obstacle occupancy or closest distance to collision for obstacles avoidance [506, 202, 713]. Overall, this chapter examines the following three questions.

**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.

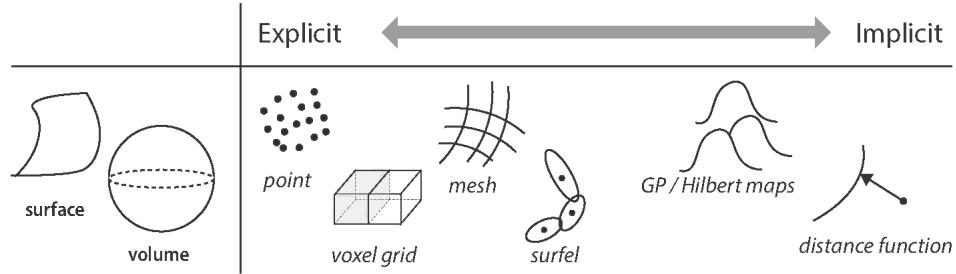


Figure 6.4 The representation can be either explicit or implicit. The illustration is simplified for clarity. In reality, the abstraction is not clearly separable and can often be applied in a combined manner.

**Q2. How should the world be represented?** This is the question of what space abstraction we use for representation. Broadly, 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 6.4.

**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.

### 6.2.1 Occupancy Maps

Since their introduction over three decades ago by Elfes and Moravec [202, 506], 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 [539]). 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.

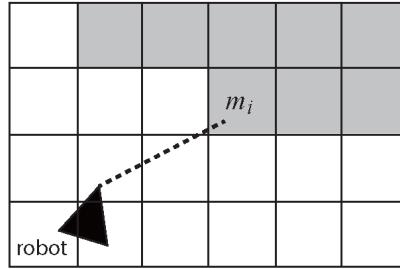


Figure 6.5 A simple ray-casting example in 2D. Given a range measurement at a certain (computed or estimated) azimuth, the return of the range measurement indicates the existence of an obstacle.

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 6.5. 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 [506],

$$l(m_k|\mathbf{z}_{1:t}) = l(m_k|\mathbf{z}_{1:t} - 1) + l(m_k|z_t), \quad (6.6)$$

where  $l(\cdot|\cdot) = \log(o(\cdot|\cdot))$ , and  $o(\cdot|\cdot)$  is the odds form:

$$o(m_k|\mathbf{z}_{1:t}) = \frac{p(m_k|\mathbf{z}_{1:t})}{1 - p(m_k|\mathbf{z}_{1:t})}. \quad (6.7)$$

The main advantage of occupancy mapping is shown in (6.6), 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 disregard spatial correlations that 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.

### 6.2.2 Distance Fields

Another way of representing the geometry surrounding the robot 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 6.6 illustrates how this implicit function values are determined.

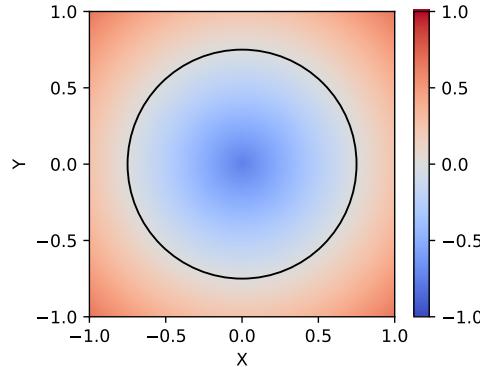


Figure 6.6 A solid 2D disk represented as an explicit surface (black outline) and implicit surface (colored field). The implicit surface function is negative inside the object (blue) and positive outside (red). Note how the function's zero crossings correspond to the surface itself.

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 [152] and popularized in the field of robotics by KinectFusion [526], 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 6.4.4.

### 6.2.3 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.

## 6.3 Map Representations

### 6.3.1 Explicitness of Target Spatial Structures

As summarized in Figure 6.4, 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 a 2D manifold, 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 6.4.3), which comprises vertices, edges, and faces. These meshes have the ability to encode the directed surfaces of a volume by forming connected closed polygons, more commonly triangles. Surfels (Section 6.4.2) are also popular abstraction 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 computed 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 6.4.4) 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. GP (Section 6.4.5) and Hilbert maps (Section 6.4.6) are well-known examples of implicit representations.

### **6.3.2 Types of Spatial Abstractions**

#### *6.3.2.1 Points*

Given range measurements, a straight-forward dense map representation is using 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 [817, 165, 728] use efficient nearest neighbor search, such as voxel grids or hierarchical tree-based representations (see Section 6.3.3), to subsample and store the point clouds, *e.g.*, store only a limited number of points per voxel [165, 728] or only specific points that meet a certain criterion are stored [817]. 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 [726] or a Signed Distance Function (SDF) via marching cubes [727]. 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.

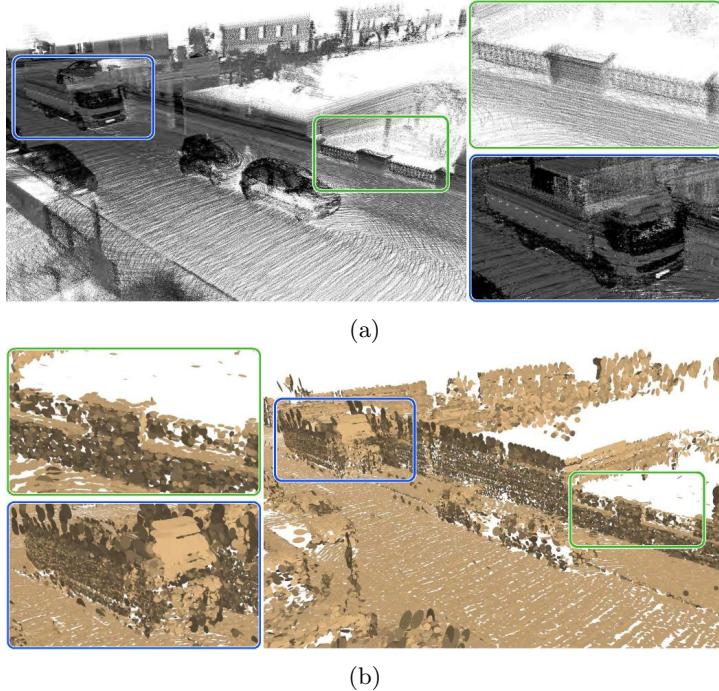


Figure 6.7 Qualitative comparison of maps generated by accumulating point clouds and surfels from a sequence of LiDAR scans from the KITTI dataset [253] Sequence 07. (a) Point cloud map. The brightness of points indicates the remission of the LiDAR measurements. (b) Corresponding surfel map based on circular disks. The complete map with all accumulated point clouds use 2.95 GB, while the corresponding surfel map by SuMa [50] uses only 160 MB.

### 6.3.2.2 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 [570], we can encode such information by adding directional information to a point. Surfels are commonly represented via circular or elliptic discs [363, 50, 755, 72, 71], or more generally ellipsoids [670, 671, 182, 850] modeled with a Gaussian. So-called splatting [850, 71] allows to integrate texture information but also blend overlapping surfels into coherent renderings of a specific viewpoint.

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  $r \in \mathbb{R}$ . As rendering primitive such surface patches can be efficiently rendered using the capabilities of modern graphics processing units (GPUs), which can be exploited to efficiently

render arbitrary views. This accelerates point-to-surfel associations and leads to the substantial memory reduction.

Figure 6.7 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) [58, 667]. 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.

#### 6.3.2.3 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 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.

### 6.3.2.4 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.

### 6.3.2.5 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 [708] and planes in 3D [355, 254, 708] 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 [603]) or non-parametric (*e.g.*, GP-based) have the ability to model the environment with fewer assumptions, still in a continuous manner. From occupancy [539], implicit surface [759, 420], distance fields [764], and surface itself [722], 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 [259].

A key advantage of the continuous functions for mapping is that if they are chosen

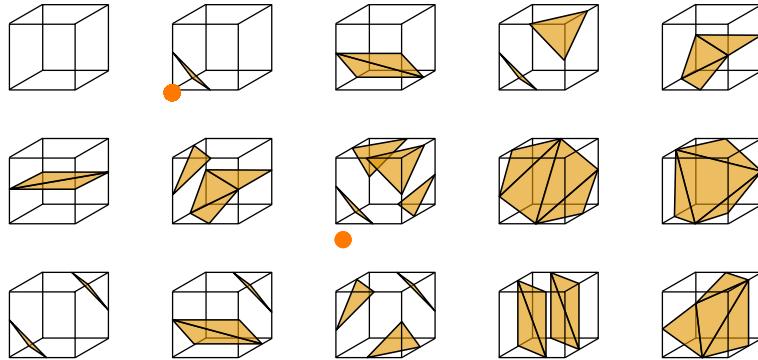


Figure 6.8 The algorithm works by projecting the cubes into the implicit surface, querying the sign of the values at the corners of the cubes, and looking up which one of the 15 configurations these values map to.

to be at least once differentiable they will be able to provide gradients. Gradient information can be key for localisation to compute surface normals [765], loop closure to compute terrain features [258], for data fusion [420] and planning [765] applications.

#### 6.3.2.6 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 [639].

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 [458]. 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 6.8). The positions of their vertices are then refined through linear interpolation. Meshes, including those from BIM or CAD models, can, in turn, be sampled to create points or surfels.

Occasionally, a discrete representation must be converted into a continuous one. This is typically achieved by solving an optimization problem over the parameters

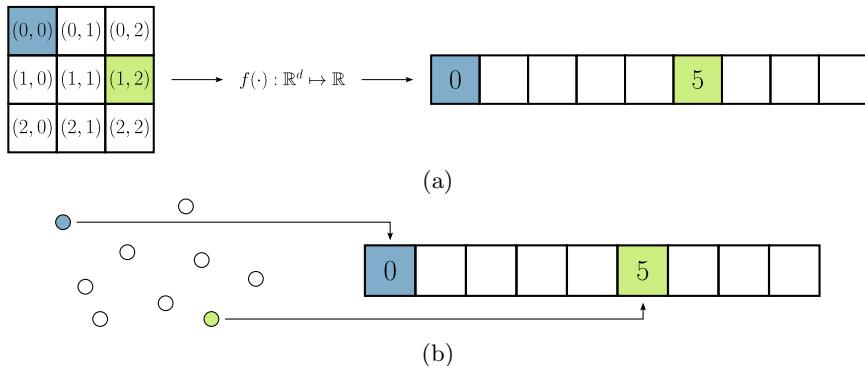


Figure 6.9 (a) Mapping of logical grid coordinates to an underlying naive array-based storage through a function  $f(\cdot)$  that uniquely maps coordinates to linear indices. (b) Storage of unordered data directly into an array structure.

of the continuous representation, minimizing the fitting error with respect to the discrete data.

### 6.3.3 Data Structures and Storage

Previously introduced abstractions 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.

#### 6.3.3.1 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 6.9.

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.

#### 6.3.3.2 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.

#### 6.3.3.3 Tree-based Data Structures

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 [53], bounding volume hierarchies (BVH) [145], and octrees [494].

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 6.10. 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.

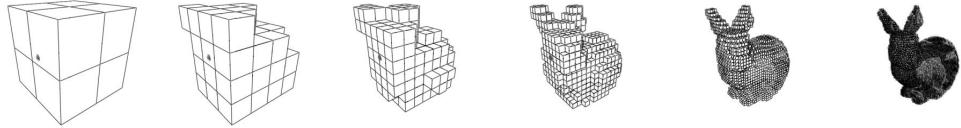


Figure 6.10 Example of an octree and its octants at different levels of the tree hierarchy. Each level of an octree subdivides the space in more fine-grained octants. Note that deeper levels of the octree only represent the occupied space.

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 [203, 541, 52].

#### 6.3.3.4 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 [530] 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 voxel grids, the VDB<sup>3</sup> data structure [523, 522] splits the space into hashed blocks, but stores a hierarchical tree inside each block. This data structure provides all the

<sup>3</sup> VDB refers to sparse volumetric data and stands for several different things as Voxel Data Base or Volumetric Data Blocks. Here we follow the terminology used in [523].

Section	Space Abstraction Type	Representing Map Entities
6.4.1	Points	Surface
6.4.2	Surfels	Surface
6.4.3	Mesh	Surface (connected)
6.4.4	Voxels	Occupancy or Implicit surface
6.4.5 - 6.4.6	Continuous function	Occupancy or Implicit surface

Table 6.1 *Summary of presented mapping methods.*

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.

## 6.4 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 6.1. 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.

### 6.4.1 Points

As mentioned in Section 6.3.2.1, 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 [817, 165, 728] 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 [817], filter the raw point clouds to extract corner and surface points thereby significantly reducing the point cloud size. A voxel grid 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 [642, 734, 553, 442, 580, 641] refine the extraction of points [642, 734, 553], improve the optimization pipeline [553, 442], or integrate information of an IMU [580, 641].

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 [165] 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.* [165] and Vizzo *et al.* [728] 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 6.3.2.1, 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.

#### 6.4.2 Surfels

For surfels, similar strategies can be applied as for point clouds, but notably Stückler *et al.* [670] and follow-up work by Dröschel *et al.* [182] 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.

In contrast, Whelan *et al.* [755] 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 [363]. A key contribution of Whelan *et al.* is leveraging a map deformation that directly deforms the surfels instead of relying on a pose graph optimization, 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.* [556].

Similarly, Behley *et al.* [50] 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 ElasticFusiun [755], the approach relies on pose graph optimization 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 [739].

#### 6.4.3 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 6.3.2.6).

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 [105]. 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 6.2.2, 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 [320], such methods may yield implicit surfaces that are discontinuous. Tackling this issue, Carr et al. [103] 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. [646] overcome this limitation by locally approximating the input points using moving least squares (MLS). Going one step further, Poisson Surface Reconstruction [360] 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.

#### 6.4.4 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.

##### 6.4.4.1 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 6.2.1) or a *distance* metric (see Section 6.2.2). The previous discussion in Section 6.2.3 can be used to decide between the two.

Since the introduction of the original continuous probabilistic occupancy measurement model for sonar [506], simplified piecewise-constant models have been developed to reduce computational costs [322]. 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. [457] 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 6.2.3).

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 6.2.2), 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 [89]. 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 [542] popularized incrementally building ESDFs. Voxblox fuses the sensor data into a TSDF and then updates its ESDF using a brushfire algorithm [413]. Subsequently, FIESTA [285] proposed a hybrid approach that incrementally updates an ESDF map from an occupancy map instead.

#### *6.4.4.2 Methods by Data Structure*

The simplest data structure for volumetric mapping is a static 3D array. As shown by KinectFusion [526], 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 proposed a voxel-block hashing scheme [530], 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 [285]. 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 [322] 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 [723], 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 [523] data structure was first introduced for the visual effects (VFX) industry and subsequently used by several volumetric mapping frameworks [478, 727]. As discussed in Section 6.3.3.4, 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 [268, 608] or an ESDF [542, 285] 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.

#### *6.4.4.3 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.

#### *6.4.4.4 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 [322] 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 [187] or multi-resolution projective integration [723]. Supereight2 [234] 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 [598] 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.

#### 6.4.5 GPs

As mentioned in Section 6.3.2.5, 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. GP [590] 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 3.2 how GP can be used for continuous time trajectory representation. As will be apparent, GP 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 [722, 539, 258, 384].

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

$$f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{m}(\mathbf{x}), \mathcal{K}(\mathbf{x}, \mathbf{x}')). \quad (6.8)$$

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,

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\left(\mathbf{1}, \begin{bmatrix} \mathbf{K}(\mathbb{X}, \mathbb{X}) + \sigma_j^2 \mathbf{I} & \mathbf{K}(\mathbb{X}, \mathbb{X}^*) \\ \mathbf{K}(\mathbb{X}^*, \mathbb{X}) & \mathbf{K}(\mathbb{X}^*, \mathbb{X}^*) \end{bmatrix}\right), \quad (6.9)$$

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,

$$\bar{\mathbf{f}}_* = \mathbf{K}(\mathbb{X}^*, \mathbb{X}) [\mathbf{K}(\mathbb{X}, \mathbb{X}) + \sigma_j^2 \mathbf{I}]^{-1} \mathbf{y}, \quad (6.10)$$

and the covariance equation is,

$$\text{cov}(\mathbf{f}_*) = \mathbf{K}(\mathbf{X}^*, \mathbf{X}^*) - \mathbf{K}(\mathbf{X}^*, \mathbf{X}) [\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_j^2 \mathbf{I}]^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}^*). \quad (6.11)$$

Here, (6.10) and (6.11) 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 [539] 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 (6.9). After the regression is performed using (6.10) and (6.11), 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 [384, 385, 738, 259].

A key advantage of mapping with GP-based functions is that the estimated quantity can be linearly operated [625] 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 [259, 448]. 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.

#### 6.4.5.1 Gaussian Process Implicit Surface

Implicit surfaces can also be represented by a GP. Gaussian process implicit surface (GPIS) techniques [759, 485, 448, 336] 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 [386, 666, 420, 764].

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 [625]. 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):

$$\begin{bmatrix} \mathbf{d} \\ \nabla \mathbf{d} \end{bmatrix} \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}')). \quad (6.12)$$

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 [485, 336]. 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 [764, 765, 418].

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.

#### **6.4.6 Hilbert Maps**

Hilbert Map (HM)s [588] are in many ways similar to GPOMs [539]. 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 [758], Random Fourier Features [585], and Sparse Kernel [496], 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 6.11, is a circle or a sphere and their values need to be combined to reconstruct intricate details of the environment. Therefore there

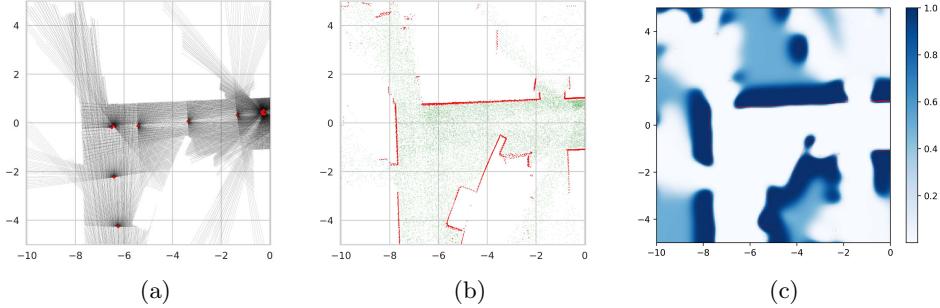


Figure 6.11 (a) The observations by a robot using a 2D LiDAR sensor, which is turned into a dataset (b) Free (green) and occupied (red) points. These are then used to train a Hilbert Map as seen in (c).

is a tradeoff in the form of number of kernels and their lengthscale affecting the computational cost, reconstruction detail, and interpolation ability.

#### 6.4.7 Deep Learning in Mapping

With the recent interest of the computer vision and robotics community in novel view synthesis using neural radiance fields (NeRFs) [500], which provided 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 location leads to a continuous representation that can be turned into meshes at arbitrary resolutions, but also can lead to more complete representation due to the interpolation capabilities of the learned function.

Similar to implicit representations, the 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 recently also an interest in incremental approaches [676, 833] and approaches that estimate poses on-the-fly using a neural representation [676, 624].

In particular, the approach of Sucar *et al.* [676] 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 [565, 844], octrees [833], points [624, 169], 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 to decouple 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 currently rapidly evolving and integrating ideas from classical representation, such as surfel splatting [362, 489], to achieve remarkable results in terms of reconstruction quality, but also capabilities. 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 [850, 71] have gained considerable popularity, demonstrating significant potential in various SLAM-related works. These will be further detailed in Chapter 18.

## 6.5 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 6.4.

### 6.5.1 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. Most existing mapping frameworks can be grouped into one of three categories of approaches. 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.

### 6.5.2 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 6.2, 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 coordinates. The exact opposite is true for implicit representations, which are therefore commonly used for collision checking tasks.

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 6.2 *Complementary strengths and weaknesses of explicit and implicit surface representations.*

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 6.4.3), 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.

### **6.5.3 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 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 [366] – an explicit learning-based representation bearing similarities to surfels – lead to an increasing interest into approaches using splatting [362, 489, 815]. 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 suspect that learning-based methods can increase the generality and expressiveness of dense representation, while improving their ability to handle noisy measurements, incomplete observations and dynamic objects through learned priors.

# 7

## Certifiably Optimal Solvers and Theoretical Properties of SLAM

Author I, Author II, and Author III

## **PART TWO**

### SLAM IN PRACTICE



# 8

## Prelude

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

Part I laid the foundations 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 preprocessing, time synchronization, noise filtering, and measurement modeling.

This part explores the widely adopted sensors in SLAM, organizing the chapters 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.

### 8.1 Structure of SLAM Framework

Most SLAM front-ends rely on two key modules: odometry and loop closure. While these modules are essential, priors can be occasionally incorporated alongside them. The *odometry* module imposes measurement constraints between consecutive nodes in a graph, typically arranged in sequential time order. By chaining together these odometry estimations, 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, a unary factor may also be included as a prior, depending on the sensor type. These unary factors are often used to incorporate additional sensor data or priors, enhancing the overall SLAM system's

accuracy and robustness (*e.g.*, GPS and depth). Although not the central element, these factors serve as a valuable supporting component.

### **8.1.1 Odometry**

The odometry module estimates the relative transformation between two consecutive nodes in a 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 loss. Many visual and LiDAR SLAM systems compute odometry in this manner.

At times, the odometry computation involves more detailed procedures. When kinematic or dynamic information is available, such as from radar or leg encoders and contact sensors, the odometry computation becomes less relative. For instance, in radar, both range and radial velocity are available. By using the radial velocity to infer ego-velocity, a 6-DOF factor can be integrated between two frames. Leg odometry can also be enhanced by incorporating data from encoders or contact sensors, along with robot kinematics, during the odometry computation. The most widely used odometry method is inertial odometry. Inertial measurements involve more complex computations between two nodes, often implemented as a pre-integration factor. This pre-integration can serve as odometry but is frequently combined with other sensors to form a more comprehensive pre-integration factor.

Its naming varies based on the primary sensor used, such as visual odometry, LiDAR odometry, or leg odometry. In this part, each chapter provides a detailed exploration of the odometry module tailored to specific sensor modalities.

### **8.1.2 Loop-closure**

The loop-closure module entails two problems. The first problem is *place recognition*, which involves using information-retrieval methods to identify a candidate loop closure in a topological manner. In this book, we refer to place recognition as the loop-closure candidate-detection problem. It is the retrieval task that involves identifying a query's nearest index from the database using retrieval or matching algorithms. We will also use the terms *query* and *database*, which are commonly used in the information-retrieval literature, but also commonly used in place recognition. The query is the current sensor measurement and the database includes a collection of places in the form of raw sensor measurements or descriptors.

Loop-closure detection is commonly performed using a variety of extrovert 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, introducing visual words to implement a bag-of-words model. Primarily, binary bag-of-words (DBoW) 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, 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, avoiding 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 secured, the follow-up registration encompasses estimating a relative transformation (either full 6-DOF or partial) between the query node and the candidate node, resulting in a binary factor.

### 8.1.3 Priors and Unary Factors

Some sensors provide measurements as priors rather than relative measurements between frames. This is the case for sensors like GPS measurements in terrain navigation, depth sensors in underwater environments, or radar providing instantaneous velocity. These priors can be incorporated into the SLAM system as unary factors, constraining a node with details and enhancing the accuracy of the overall system. Unfortunately, their modeling will not be discussed in detail in the following chapters. For further insights, readers are referred to dedicated resources on GNSS [99] or UWB applications.

## 8.2 Sensors in a Factor Graph

From the factor-graph SLAM perspective, each sensor modality produces a factor, acting as a building block for the entire SLAM system. An example factor graph in Figure 8.1 illustrates a possible integration case. 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 through integration.

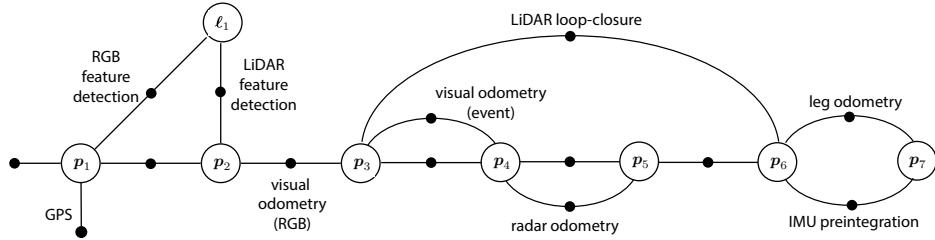


Figure 8.1 Sample factor-graph consists of all sensor modalities introduced in this part.

### 8.2.1 Selecting the Right Sensor for Your Application

Let's briefly overview the extrovert perceptual 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 their close 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, and shadows, and their performance is significantly limited in extreme environments, such as those with fog or smoke.

**LiDAR:** LiDAR directly measures distance without the need for triangulation, enabling it 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, LiDAR's main drawback is its high cost, which can be a burden in terms of both price, memory, power consumption, and computation. The point cloud data it generates is often extensive, requiring substantial computational power and memory for processing and storage. These challenges should be carefully considered when using LiDAR in budget-sensitive, real-time systems (*e.g.*, drones). Furthermore, despite their robustness to the 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 completing SLAM in extreme environments where LiDAR and cameras may be limited. However, radar data tends to be noisy and extremely sparse, which can complicate its interpretation. 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) while consuming very low power. By transmitting data only for pixels that change, they are highly energy-efficient, capturing 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 (HDR) 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.

### 8.2.2 Sensor Fusion

In SLAM literature, the common practice is to combine multiple sensors in a complimentary manner. The most favored sensor in this integration would be inertial measurements. Incorporating an Inertial Measurement Unit (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 VINS, LIO, and RIO. These systems leverage the pre-integration of IMU data to maintain continuous tracking and correct drift over time. While IMU-only systems have seen some development, such as efforts in IMU-based odometry, 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 UGVs, 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.

We also need to consider the heterogeneous aspect during sensor fusion. Camera and LiDAR combination would be one of the popular SLAM system. This is

because semantic information from camera and direct range measurement from LiDAR can effectively enhance each sensor's limitation. Camera-radar fusion follows a similar strategy, complementing each other to mutually enhance performance and address limitations. 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 among 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.

### ***8.2.3 Calibration and Synchronization of Sensors***

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 solves for 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 the multi-modal registration problem. Due to differences in data formats and underlying physics, comparing data from two sensors is often difficult, much like comparing apples to oranges.

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 in real-time by treating it as an optimization parameter.

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 the IMU—may be necessary to bridge the gaps between sensor data streams.

### 8.3 How to Read this Part?

The organization of this part is structured around different sensor modalities. Each chapter follows a similar structure, focusing on odometry, place recognition, and SLAM details for each sensor. It is recommended to start with the Visual SLAM chapter to grasp the basic definitions of each SLAM module. After that, the chapters can be read in any order.

Following the discussion on RGB cameras SLAM in Chapter 9, two range sensors are introduced: LiDAR SLAM in Chapter 10 and radar SLAM in Chapter 11. Moving beyond conventional sensors, we delve into event camera SLAM in Chapter 12. The IMU, covered in Chapter 13, serves as an inertial odometry sensor but demonstrates greater potential when combined with other modalities through preintegration techniques. Additionally, this part explores how odometry can be modeled for legged robot systems in Chapter 14.

# 9

## 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, or short visual SLAM. With cameras being omni-present, cheap and power-efficient, the potential of visual SLAM for autonomous robots, for self-driving cars or for mixed and augmented reality is sheer endless.

### 9.1 Historic Background and Terminology

#### 9.1.1 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 of aerial imagery for photogrammetric reconstruction of glaciers in the Alps and also advocated techniques of projective geometry [241]. His doctoral student Otto von Gruber formalized the mathematical framework of bundle adjustment 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 bundle adjustment and teamed up with Ameri-

can Duane C. Brown in the 1950s to deploy these methods on the largest computers of their time.

Reconstruction methods and the field of structure from motion 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 [705] 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 multiview 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 bundle adjustment 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 [461] or SURF [47], and more recently a multitude of learning-based descriptors. In order to cope with incorrect point correspondences researchers developed sampling strategies such as RANSAC [221] that allowed the method to revisit the correspondence estimation in alternation with model fitting.

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 realtime reconstruction from a moving camera. A prerequisite for such online and realtime approaches was therefore the development of causal methods such as [137] 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 [349, 156].

### 9.1.2 Terminology

The following terms are often used interchangeably to describe similar processes; however, place different emphasis depending on application and community they are used in.

**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 bundle adjustment*, 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 revert to bundle adjustment for optimization. Landmark projects, such as *Building Rome in a Day* [18], 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 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.

## 9.2 Visual SLAM Fundamentals

Before we go into detail on visual SLAM let us introduce a few fundamentals.

### 9.2.1 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 **photo-**



Figure 9.1 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 [184], the Kannala-Brandt (KB) model [358] and the Double Sphere (DS) model [719]. 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.

**metric component**, which describes how physical light intensity (radiance) maps to pixel values.

#### 9.2.1.1 Geometric Camera Models

##### Perspective Cameras

The projection function is generically referred to as:

$$\mathbf{z} = \pi(\mathbf{x}^c, \boldsymbol{\xi}),$$

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 2D point in the image domain  $\Omega$ , and  $\boldsymbol{\xi} \in \mathbb{R}^n$  represents the intrinsic parameters of the camera, typically pre-calibrated 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.

In practice, there exists a wide range of projection functions suitable for different lens geometries and camera types (see some, for example, in [455]). 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  $f_u \approx f_v$ .

To account for some amount of lens distortion, the radial-tangential model is most commonly used:

$$\boldsymbol{\xi}_{RT} = [\boldsymbol{\xi}_p^\top \ k_1 \ k_2 \ p_1 \ p_2]^\top, \begin{bmatrix} x' \\ y' \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \ r = \sqrt{x'^2 + y'^2},$$

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'(1 + k_1 r^2 + k_2 r^4) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y'(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{bmatrix}, \ \pi_p(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u x'' + u_0 \\ f_v y'' + v_0 \end{bmatrix},$$

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 9.1 – up to fields of view (FOV) of  $180^\circ$ , pinhole models with a few radial distortion coefficients typically suffice. The Brown-Conrady model [184] amounts to the 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 [358] has been used, for example in [96].

$$\boldsymbol{\xi}_{KB} = [\boldsymbol{\xi}_p^\top \ \mathbf{k}_{KB}^\top]^\top, \ \pi_{KB}(\mathbf{x}^c, \boldsymbol{\xi}) = \begin{bmatrix} f_u r(\theta) \cos \psi + u_0 \\ f_v r(\theta) \sin \psi + v_0 \end{bmatrix}^\top$$

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 [719] as it offers a good compromise of accuracy and speed. In the DS 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 [719], 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.

#### 9.2.1.2 Photometric Models

The photometric calibration maps irradiance to pixel values:

$$I = f(E, T),$$

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.

#### 9.2.1.3 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.

#### 9.2.1.4 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.

## 9.2.2 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 vision-based SLAM,

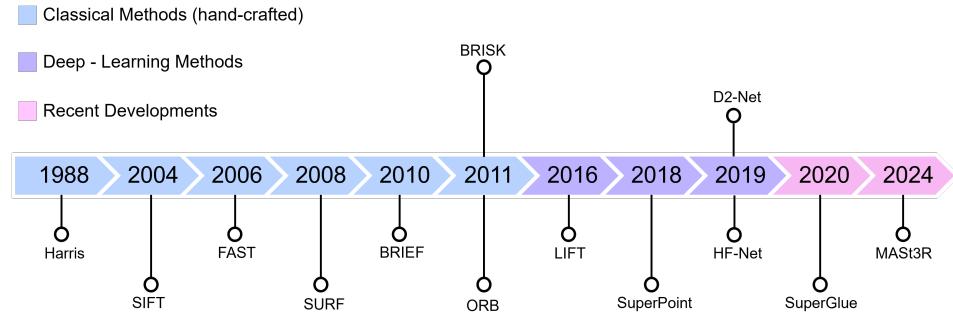


Figure 9.2 Timeline of some of the most prominent algorithms shaping the literature on visual keypoints for vision-based SLAM and image matching.

keypoint 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. 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 9.2, along with emerging trends in the field.

#### 9.2.2.1 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 [287], 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 [648] 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) [461], 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 less suitable for real-time applications.

To improve efficiency, Features from Accelerated Segment Test (FAST) [610] 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) [48] 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) [95] 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 [616] 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 [428] 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 vision-based 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.

#### *9.2.2.2 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) [802] 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 [174] 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 [626] 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 [431].

Interestingly, D2-Net [194] introduced a describe-and-detect approach that reverses 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 Structure from Motion (SfM) tasks, pushing the boundaries of deep learning-based feature extraction.

#### *9.2.2.3 Latest algorithms & trends*

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 [628] and MASt3R [425] 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 [520] 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 vision-based 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 interpretability, 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.

### 9.2.3 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:

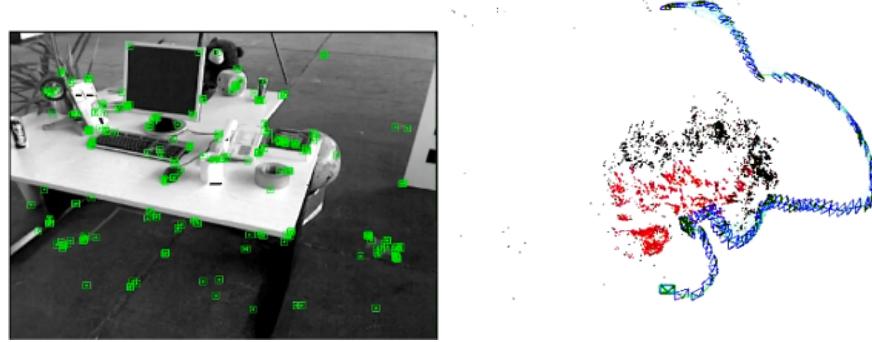


Figure 9.3 The feature-based visual SLAM problem: given a set of features matched in each image (left), estimate the position of their corresponding 3D points and the pose from where each image was acquired (right). Images generated with ORB-SLAM [518].

$$E_{\text{reproj}} = \frac{1}{2} \sum_j \|z_j - \pi(x_j^C, \xi)\|_{\Sigma_j}^2. \quad (9.1)$$

To handle outliers, robust kernels such as the Huber or Tukey kernel are applied. For example, the robust reprojection error can be expressed as:

$$E_{\text{robust}} = \frac{1}{2} \sum_j \rho \left( \|z_j - \pi(x_j^C, \xi)\|_{\Sigma_j} \right), \quad (9.2)$$

where  $\rho(\cdot)$  is a robust kernel function that reduces the influence of large residuals. In the following, for simplicity, we will drop the dependence with the camera intrinsics  $\xi$ .

#### 9.2.4 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  $C_i \in \mathbb{C}$  taken with a moving camera. To make notation simpler, we will just use the point and camera identifiers writing  $j \in \mathbb{P}$  and  $i \in \mathbb{C}$ . The goal of visual SLAM is to estimate the point coordinates  $x_j^w \in \mathbb{R}^3$  and the camera poses  $T_w^i \in \text{SE}(3)$ , both expressed in a world reference frame  $\mathcal{F}^w$ . In the monocular case, the observation of each point in each image is just the observed image coordinates  $z_{ij} \in \Omega_i \subset \mathbb{R}^2$  (Figure 9.3).

The minimization of the reprojection error, when applied to all the points and camera poses, is known as *full BA*:

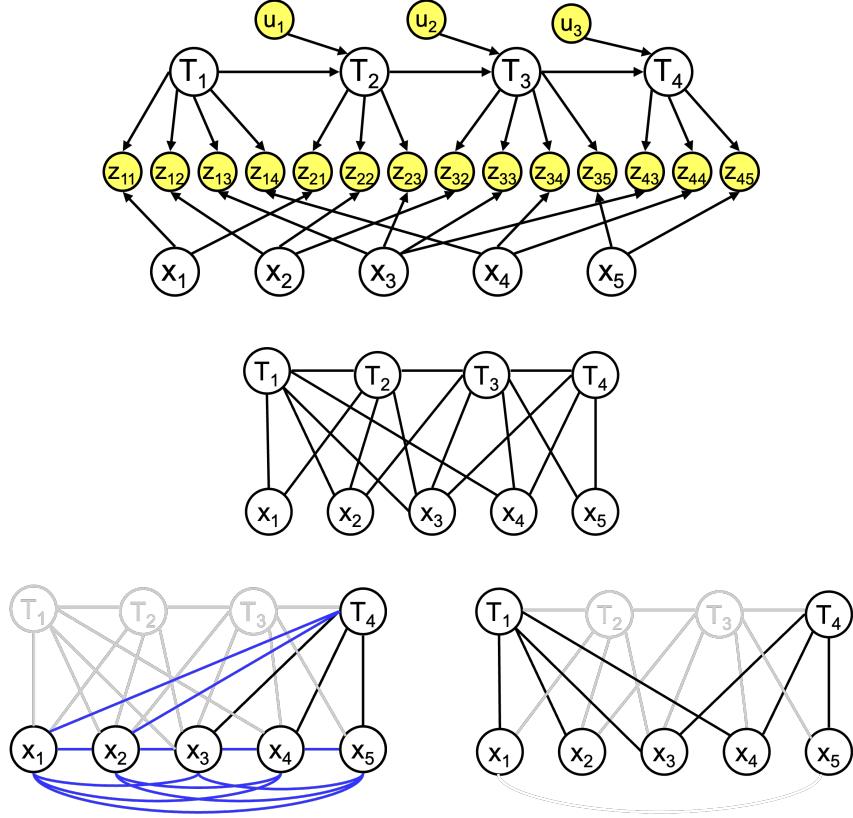


Figure 9.4 A visual SLAM example with four camera poses and five features: Bayes network (top), and its corresponding Markov random field (middle). EKF SLAM marginalizes out past camera poses, resulting in a dense graph (lower left). Keyframe SLAM keeps just a few camera poses and discards observations from intermediate images, keeping SLAM sparsity (lower right) [669].

$$\{\mathbf{T}_w^{i*}, \mathbf{x}_j^{w*} \mid i \in \mathcal{C}, j \in \mathcal{P}\} = \arg \min_{\mathbf{T}_w^i, \mathbf{x}_j^w} \frac{1}{2} \sum_{i,j} \rho \left( \|\mathbf{z}_{ij} - \pi(\mathbf{R}_w^i \mathbf{x}_j^w + \mathbf{t}_w^i)\|_{\Sigma_{ij}} \right). \quad (9.3)$$

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 (9.3). 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.

- **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 9.4), limiting them to a few hundred features. Also, filtering methods do not re-linearize past observations, losing accuracy.
- **Keyframe-Based Approaches:** They simplify the problem by keeping just a few images, called keyframes [389]. 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 [669].
- **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.

#### **9.2.5 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.

#### **9.2.6 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 [247] 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 [33, 339].

#### **9.2.7 Initialization**

The minimization of the reprojection error of Equation 9.3 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.

### ***9.2.8 Common Steps***

- 1 Feature Matching: The system matches current observations to landmarks or keypoints stored in the map. Efficient descriptor matching techniques, such as ORB or SuperPoint, are typically used.
- 2 Pose Estimation: Using the matched features, the system estimates the camera's pose relative to the map, often leveraging robust optimization techniques to account for outliers.
- 3 Validation: The estimated pose is validated by checking consistency with the map and potentially re-optimizing if mismatches occur.

### ***9.2.9 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.

## **9.3 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, the problem of visual SLAM can be tackled in several stages. 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 problem 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.

### ***9.3.1 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 capture the visual odometry: 1. 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 bundle adjustment. 2. 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 bundle adjustment.

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.

### 9.3.2 Mapping Back-End

The back-end optimizes the trajectory and map using global optimization techniques like bundle adjustment or factor graph solvers. This step refines the estimates provided by the front-end and integrates observations into a consistent map. As a consequence, one obtains more long-term consistency, and long-range distortions are reduced.

### 9.3.3 Visual Place Recognition and Relocalization

Visual odometry is prone to drift because errors in the camera tracking will aggregate over time. In the absence of additional sensors like IMUs, wheel odometry or 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.

### 9.3.4 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.

### 9.3.5 Keypoint-based Image Alignment

Although full BA (equation 9.3) 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 [389]:

- 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*:

$$\mathbf{T}_w^i = \arg \min_{\mathbf{T}_w^i} \sum_{j \in P} \rho \left( \| \mathbf{z}_{ij} - \pi(\mathbf{R}_w^i \mathbf{x}_j^w + \mathbf{t}_w^i) \|_{\Sigma_{ij}} \right). \quad (9.4)$$

- 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.

*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 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 [668, 518]

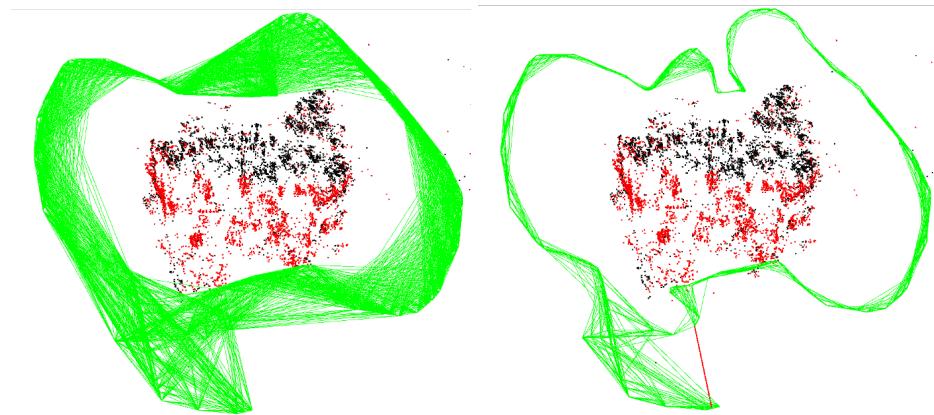


Figure 9.5 Representation of locality in ORB-SLAM [518]. The covisibility graph (left) connects keyframes that have seen at least  $\theta$  points in common (in this example  $\theta = 15$ ) and is used for local BA. The essential graph (right) is a sparser version, that in this example connects keyframes with at least  $\theta = 100$  points in common, and is used for pose-graph optimization during loop correction. ©2015 IEEE.

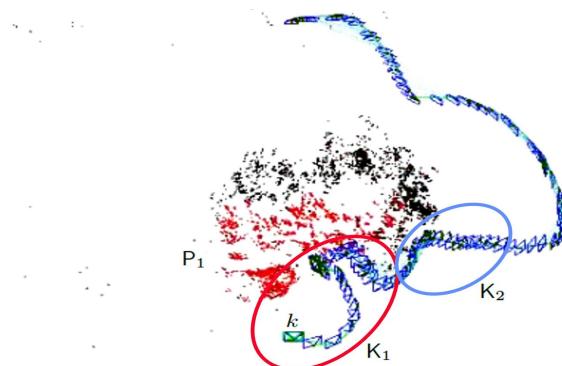


Figure 9.6 Implementation of local BA in ORB-SLAM [518]. The local map is defined by the set of keyframes  $K_1$  that contains the current keyframe  $k$  and its neighbors in the covisibility graph, and the set  $P_1$  of points seen by them (in red).  $K_2$  is the rest of keyframes in the map that see some point from  $P_1$ .

(see example in figure 9.5). In that way, *local BA* can update just a set of covisible

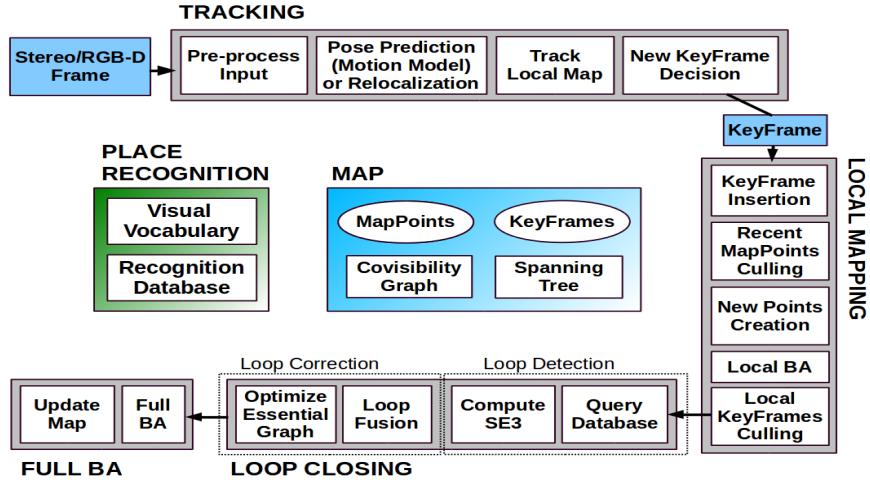


Figure 9.7 Structure of ORB-SLAM2 system [515] showing the map, the place recognition database and its complete processing pipeline composed of four threads: tracking, local mapping, loop closing and full BA. ©2017 IEEE.

keyframes and the points observed by them (figure 9.6):

$$\{\boldsymbol{T}_w^{k*}, \boldsymbol{x}_j^{w*} \mid k \in K_1, j \in P_1\} = \arg \min_{\boldsymbol{T}_w^k, \boldsymbol{x}_j^w} \frac{1}{2} \sum_{\substack{i \in K_1 \cup K_2, \\ j \in P_1}} \rho \left( \|z_{ij} - \pi(\boldsymbol{R}_w^i \boldsymbol{x}_j^w + \boldsymbol{t}_w^i)\|_{\Sigma_{ij}} \right). \quad (9.5)$$

An example of a complete visual SLAM pipeline can be seen in figure 9.7 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.

### 9.3.6 Direct Image Alignment

Direct methods such as LSD SLAM [205] or DSO [206] 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 for computing the maximum a posteriori estimate of 3D structure and camera motion given the raw sensory data. This amounts to solving the correspondence estimation

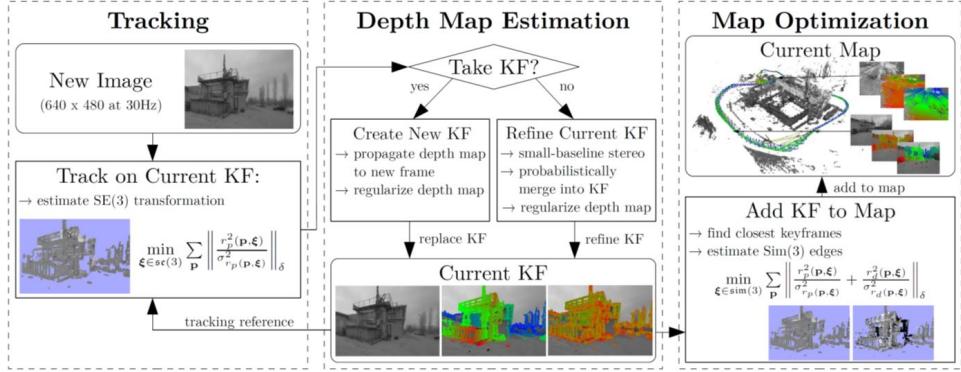


Figure 9.8 Schematic overview of Large Scale Direct (LSD) SLAM [205] showing the three components that perform direct camera tracking, direct mapping and pose graph optimization for global consistency, all running in alternation. In contrast, Direct Sparse Odometry [206] performs the optimization of 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 [206].

and SLAM problem jointly by minimizing a photometric loss of the form:

$$E_{photo} = \sum_{i \in \mathcal{F}} \sum_{z \in \mathcal{P}_i} \sum_{j \in obs(z)} \rho(I_i(z) - I_j(\omega(z, d_z, \mathbf{T}_j^i)). \quad (9.6)$$

with respect to all camera parameters  $\mathbf{T}_j^i \in \text{SO}(3)$  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 9.8, LSD SLAM optimizes for depth maps and camera motion (tracking) in alternation and performs a pose graph optimization to assure global consistency of the estimated camera motion with the computed pairwise image alignments.

In contrast, DSO [206] jointly optimizes for structure and motion in the form of a photometric bundle adjustment. 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

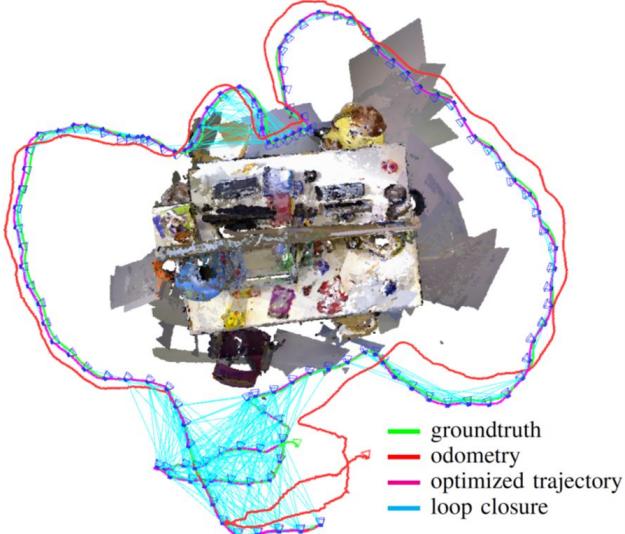


Figure 9.9 In direct visual SLAM methods, one can perform pose graph optimization in order to compute a trajectory that is globally consistent with all previously estimated pairwise image alignments [367].

precision since it relies on a statistically optimal estimate of structure and motion given all the sensory brightness data.

In realtime-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 bundle adjustment). Secondly, one can additionally run a Pose Graph Optimization (PGO) [367, 249] – see Figure 9.9 – 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 pose graph optimization called Pose Graph Bundle Adjustment (PGBA) [729] which additionally incorporates the full photometric uncertainty of bundle adjustment with the same computational efficiency (because only the camera poses are being updated).

### 9.3.7 Solving BA

Although BA could be solved using general variable elimination techniques as discussed in Chapter 2, there are specific sparsity solutions able to profit from its structure. Figure 9.10 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 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  $D$  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:

$$\begin{aligned} \mathbf{H}_{cc}^{red} &= \mathbf{H}_{cc} - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top, \\ \mathbf{H}_{cc}^{red} \mathbf{d}_c &= \mathbf{b}_c - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{b}_p, \\ \mathbf{H}_{pp} \mathbf{d}_p &= \mathbf{b}_p - \mathbf{H}_{cp}^\top \mathbf{d}_c. \end{aligned} \tag{9.7}$$

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 9.10 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.

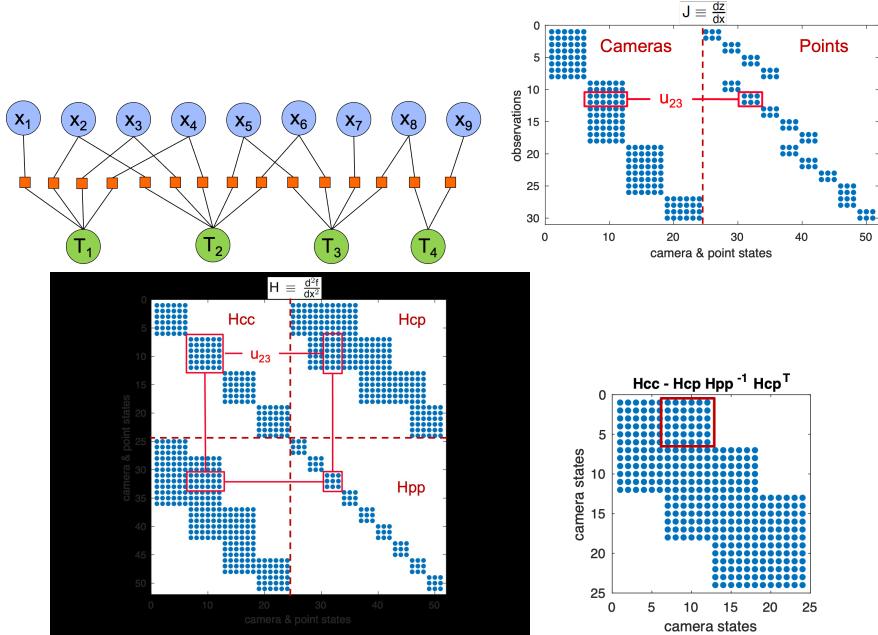


Figure 9.10 Toy example with 4 cameras and 9 points showing in the first row the factor graph and the Jacobian of point observations. The second row shows complete Hessian, and the Hessian of the reduced camera system.

### 9.3.8 Examples of Full Visual SLAM Systems

LSD-SLAM (Large-Scale Direct SLAM) [205] 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 [518, 515] 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 [96] it was extended to fisheye cameras, multimap, and visual-inertial SLAM. While originally conceived as a visual-inertial odometry system, OKVIS [426], the newest version, OKVIS2 [427], 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) [206] is a direct method for estimating 3D point cloud and camera trajectory. In contrast to LSD SLAM camera motion and landmark 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 bundle adjustment. 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 [740] and omni-directional cameras [488] have been proposed. Loop closure detection has also been added to reduce drift in longer sequences [249].

While among real-time capable approaches direct methods like DSO were shown to provide more accuracy and robustness than keypoint-based methods [206], 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 [632, 633], 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 [789]. 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 [264] where feature-based re-localization information is tightly integrated in a direct visual SLAM approach to further boost its robustness and precision.

#### 9.4 Realtime Dense Reconstruction

While traditionally bundle adjustment 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 realtime dense reconstruction from a monocular camera have been advocated over the years [675, 527, 753, 575]. Traditionally they revert to variational methods to estimate a continuous 3D structure by minimizing a loss function [675]:

$$\min_{h:\Omega \rightarrow \mathbb{R}} \frac{1}{2} \sum_{i \in \mathcal{I}(x)} \int_{\Omega} \rho_i(x, h) dx + \lambda \int |\nabla h| d^2x, \quad (9.8)$$

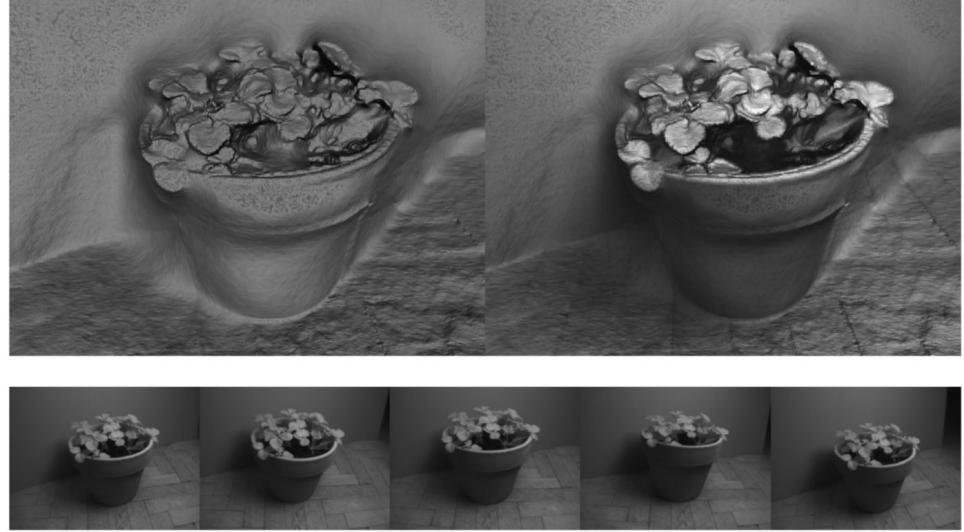


Figure 9.11 Dense reconstructions of the 3D world (above) can be computed in realtime from a handheld monocular camera (below) using variational methods that are efficiently parallelized on a GPU [675]. Related approaches were proposed in [527, 753, 575].

with respect to a dense map  $h$  that assigns a depth value to every pixel  $x$  in the image plane  $\Omega \subset \mathbb{R}^2$ . The residual

$$\rho_i(x, h) = \left| I_i(\pi T_w^i X(x, h)) - I_0(x) \right|, \quad (9.9)$$

enforces consistency of the brightness at pixel  $x$  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  $X(x, h)$ , performing the transformation  $T_w^i \in SE(3)$  to camera  $i$ , followed by a perspective projection  $\pi$  into the image  $I_i$ . Here  $\mathcal{I}(x)$  denotes the index set of the images for which the projected point lies inside the image plane. 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 9.11.

### 9.5 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

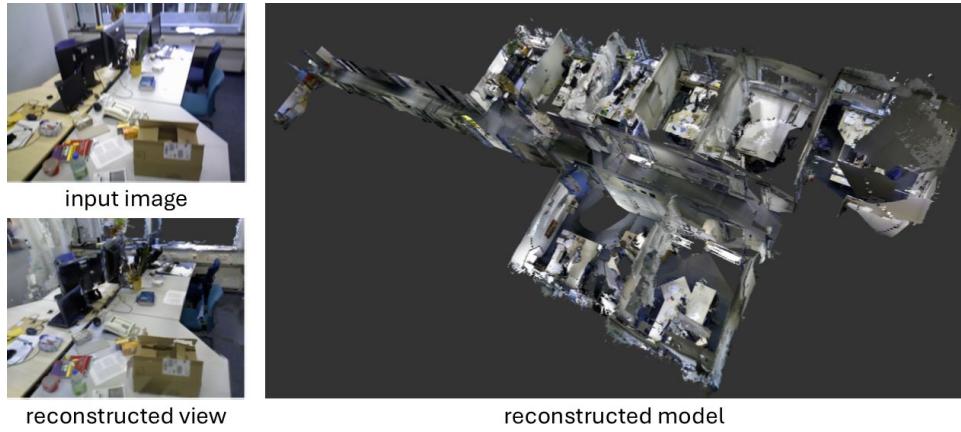


Figure 9.12 Dense reconstructions of a large-scale corridor scene with multiple offices computed from a moving RGB-D camera [663]. Using octrees [663] or voxel hashing [531] and direct camera tracking [368], such reconstructions were demonstrated to run in real-time on a GPU [663] and even on a tablet CPU [664]

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).

Kinect Fusion [528] built upon previous work for range image fusion [152] 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  $d_i(x)$  which for every voxel  $x \in V$  encodes the (signed) distance to the nearest surface point (along the viewing ray). Subsequently one can compute an aggregated distance function  $D(x)$  for all voxels as a weighted average of the individual distance functions:

$$D(x) = \frac{\sum_i \omega_i(x) d_i(x)}{\sum_i \omega_i(x)}. \quad (9.10)$$

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  $\omega_i(x)$  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 [528], or modify the above weighting scheme to ensure watertight-ness of the reconstructions [674].

While earlier RGB-D SLAM approaches track the camera by means of aligning

respective depth point clouds with the ICP algorithm [54], 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  $(I_2, d_2)$  [368]:

$$r_I(\xi) = I_2(\tau_g(x)) - I_1(x), \quad r_d = d_2(\tau_g(x)) - [g\pi^{-1}(x, d_1(x))]_z, \quad (9.11)$$

where  $\pi$  denotes the perspective projection,  $g \in \text{SE}(3)$  denotes the desired rigid body motion,  $\tau_g(x) = \pi g \pi^{-1}(x, d_1(x))$  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  $\xi \in \text{se}(3)$  [662, 367]. 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 [368] for details.

For mapping at larger scale, the uniform voxel representation is too memory-intense. Therefore one typically reverts to adaptive representations using voxel hashing [531] or octrees [663] – see Figure 9.12.

Later approaches have furthermore demonstrated scalability to larger scenes by employing a moving fusion window as in kintinous [756], or by including loop-closure as ElasticFusion [757] that deforms the entire dense map representation through a deformation graph.

## 9.6 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.

### 9.6.1 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 [204] or nano-copters [190], it is less precise than a tightly-coupled integration of sensory information.

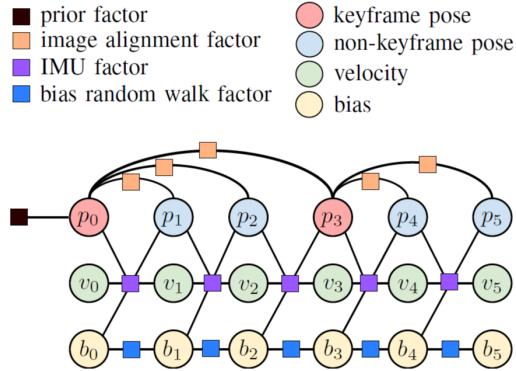


Figure 9.13 The integration of multiple sensors can be elegantly performed in tightly coupled manner using factor graphs. This is an example factor graph for tightly coupled fusion of vision and IMU [716]. It significantly increases precision and robustness of camera motion and 3D structure – see Figure 9.14.



Figure 9.14 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 9.13. The reconstructed pointclouds come from pure odometry, no loop closures were enforced [716].

First tightly-coupled approaches leverage an EKF scheme, whereby the IMU kinematics are integrated in the prediction step, and visual keypoint measurements serve as updaters—as in the seminal work of the MSCKF [508]. Alternatively, sliding-window and batch optimizers may be formulated adopting the factor graph approach discussed in detail in part 1 of this book. Figure 9.13 shows a factor graph proposed for stereo-inertial odometry [716] that combines stereo LSD SLAM with IMU information. Figure 9.14 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 [96], OKVIS2 [427], and many more.

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-

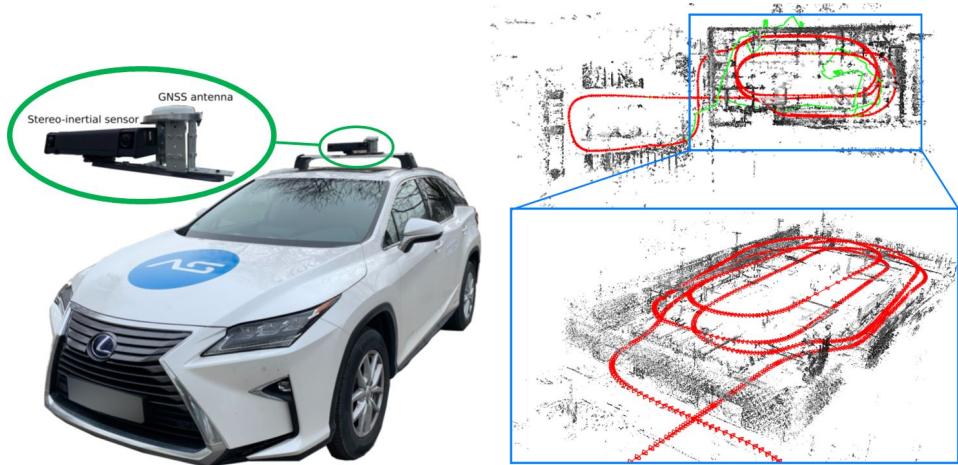


Figure 9.15 By combining multiple sensors including stereo cameras, IMU and RTK-GPS information (left), fused in a tightly coupled manner using factor graphs, one can obtain highly accurate and robust trajectories and pointclouds, both indoor and outdoor as shown in the reconstruction of a car driving on multiple levels of a parking garage (right) [754].

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 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 [516], Direct Sparse Visual-Inertial Odometry [730], VINS-Mono [583], BASALT [717] and DM-VIO [729]. 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.

### 9.6.2 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 [754, 128]—see Figure 9.15. In indoor scenarios, WiFi signals enable coarse

localization where GPS signals are unavailable, complementing visual map-based localization.

## 9.7 Bundle Adjustment Revisited

At the historical origin and at the heart of visual SLAM is the classical problem of bundle adjustment, namely, the joint estimation of all camera positions and all landmark positions. It has been studied for over a century and the classical approach detailed in Section 9.3.7 has been established and shown to work well in a multitude of seminal papers [711, 18, 630]. 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 [315, 316, 167, 168, 744, 745, 746].

The key computational bottleneck is the solution to the reduced camera system (9.7). Instead of solving this by means of an iterative conjugate gradient algorithm, Power Bundle Adjustment [745] approximates the inverse of the Schur matrix

$$\mathbf{H}_{cc}^{red} = \mathbf{H}_{cc} - \mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top, \quad (9.12)$$

by means of a matrix power series [745]:

$$(\mathbf{H}_{cc}^{red})^{-1} \approx \sum_{i=0}^m (\mathbf{H}_{cc}^{-1}\mathbf{H}_{cp}\mathbf{H}_{pp}^{-1}\mathbf{H}_{cp}^\top)^i \mathbf{H}_{cc}^{-1}. \quad (9.13)$$

This power series provably converges to the true inverse for increasing cut off parameter  $m$  [745]. The main advantage is that tedious matrix inversion is replaced by simple matrix multiplications, which can be done much faster and more memory-efficiently.

The dependency on initialization is alleviated in [315, 316] by reverting to the concept of variable projection—see Figure 9.16. To this end, the bundle adjustment 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 [746] offers a scalable solution for large-scale bundle adjustment problems without initialization.

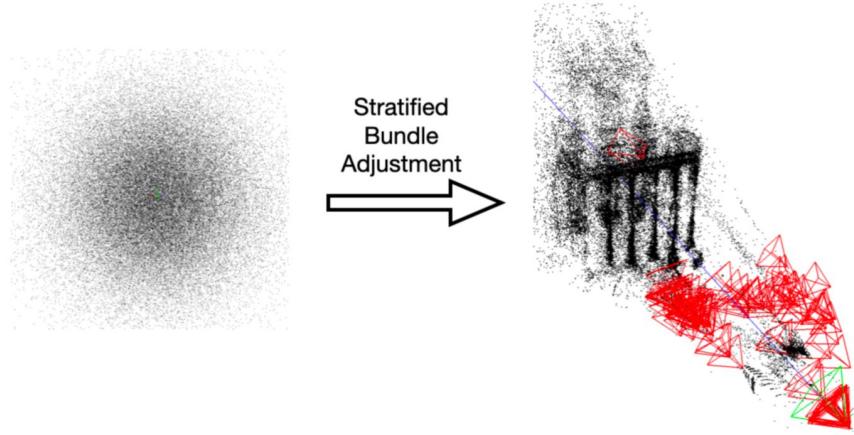


Figure 9.16 In a series of papers [315, 316, 745, 746], researchers advocate the use of variable projection methods and matrix power series in order to solve large scale bundle adjustment problems without initialization in a runtime- and memory efficient way. As shown above, camera poses and landmarks can thus be computed starting from a random initialization.

### 9.8 Recent Developments

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. Moreover, one should emphasize that there have been many exciting developments 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, bundle adjustment, dense reconstruction, etc.) are being enhanced or replaced by learning-based formulations. All these ideas will be discussed in more detail in Part 3 of this book.

# 10

## 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 major sensing modalities used in robotics and computer vision. LiDAR is a technology which 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 [159, 451]. 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 [333]. In the late 1990s, researchers began exploring the use of LiDAR for real-time SLAM in robotics. In the next section we will quickly review the underlying technology within different types of LiDAR sensors.

### 10.1 LiDAR Sensing Preliminary 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 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) [397] and therefore the accuracy and frequency of measurements. Besides TOF, the techniques of Amplitude Modulated Continuous Wave (AMCW) and Frequency

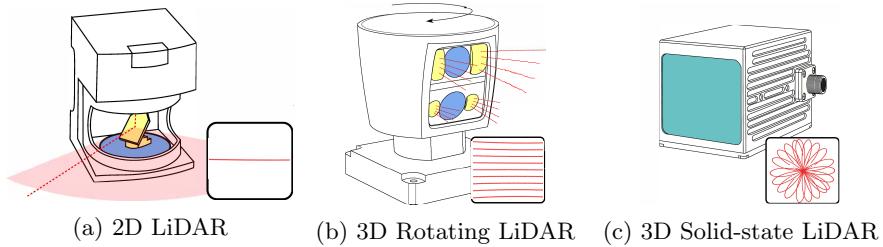


Figure 10.1 Common LiDAR sensor types and their beam patterns. (a) A standard 2D LiDAR sensor with a rotating transmitter mirror (yellow). The encoder disk (blue) is used to measure the rotation angle of the mirror. (b) An example of a mechanical multi-beam LiDAR emitting multiple laser beams from the 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) An solid-state 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.

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 [606] and can be classified into mechanical LiDARs, scanning solid-state LiDARs, and flash LiDARs. Among these, we will examine two commonly utilized types: 2D/3D mechanical LiDARs and solid-state 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 10.1a. 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 DARPA Grand Challenges (an early self-driving car competition) in the 2000's and lead to the development of the pioneering Velodyne HDL-64E sensor. It was used by most of the teams [715, 505, 357] 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 10.1(b). The resulting point cloud can capture a highly detailed 3D depiction of objects and the sensor's surroundings as shown in Figure 10.2, as discussed in various studies [752, 370, 606]. Subsequently, the technology has evolved — with the size and price of LiDAR sensors dropping significantly.

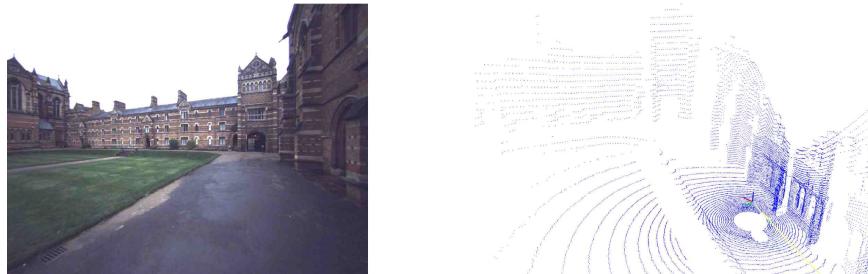


Figure 10.2 Example of a single multi-beam LiDAR scan and a corresponding image. The scan is from a 64-beam Hesai QT64 scanner with 104° vertical field-of-view. Note the individual scanning lines and also how dense the point cloud is close to the device (the colored axis in the 3D view) and much sparser the cloud is further away.

Scanning solid-state LiDAR systems have emerged more recently. In contrast these sensors use Micro-electromechanical (MEMS) [314] mirror technology or optical phased arrays (OPA) [300] to avoid mechanical rotation, leading to more robust and efficient operation. A notable advancement is the use of Risley prisms [443] in scanning solid-state LiDAR, which enables rapid, controlled beam steering without physical movement. This innovation results in a more compact system, albeit with more limited FOV. 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.

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 LiDAR continuously projects 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 [769], 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 for different applications.

## 10.2 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. The well-known ICP algorithm [55, 619] is a fundamental technique for point cloud registration and it can be used to determine this relative transformation. 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 [464], 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 [546, 396]. While points and lines are common choices in 2D scan matching, Olson [545] 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 [288] or rotating manner [68], or by passively using the motion of a human carrier [70] or vehicle [68] 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 [817] demonstrated real-time scan matching capabilities forming the basis for follow-up methods in LiDAR odometry and SLAM.

### 10.2.1 Foundations of Scan Registration

Scan registration is a fundamental component of 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  $\mathbf{R} \in \Re^{3 \times 3}$  and translation  $\mathbf{t} \in \Re^3$ , that can best bring one of the scans (potentially a recent scan from a sensor) into alignment with the other (*e.g.*, a 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 6, 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 [55] 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

$$\mathbf{R}^k, \mathbf{t}^k = \arg \min_{\mathbf{R}, \mathbf{t}} \sum_{(\mathbf{p}, \mathbf{q}) \in \mathcal{C}} d(\mathbf{p}_i, \mathbf{R}\mathbf{q}_i + \mathbf{t}), \quad (10.1)$$

where the set of correspondences  $\mathcal{C}$  between the source point cloud  $\mathcal{P}$  and target point cloud  $\mathcal{Q}$  is given by

$$\mathcal{C} = \{(\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in \mathcal{P}, \mathbf{q} \in \mathcal{Q}\}. \quad (10.2)$$

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 (10.1), two components must be specified:

- 1 While geometric relation is used defining 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.
- 2 How is the correspondence set  $\mathcal{C}$ , used for minimization, determined? This involves identifying the corresponding target point  $q \in \mathcal{Q}$  for each source point  $p \in \mathcal{P}$ .

Common approaches used these two components will be discussed in the following sections.

#### 10.2.1.1 Distance Measure in Registration Residual

The first component involves deciding which geometric elements to use for defining the residual. Typically, points, lines, and planes are the most commonly used geometric elements, as summarized in Figure 10.3.

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 [825] and Besl and McKay [55] formulated shape (*e.g.*, curves

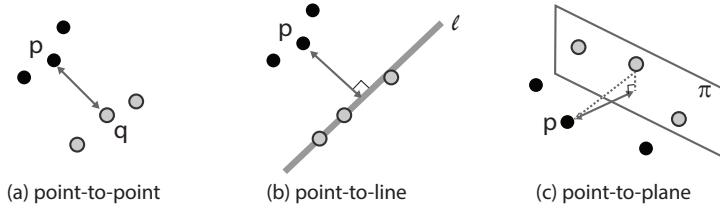


Figure 10.3 Typical distance metrics used in ICP. (a) Point-to-point distance is as straightforward as the Euclidean distance between two points. (b) and (c) The point-to-higher-level feature (*e.g.*, line or plane) is calculated as the shortest distance to the reconstructed line or plane using the target points.

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 and outliers. Distances to lines are also 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.

Extending from these basic geometries, other ICP variants introduce different distance metrics. Techniques which employ multi-distance metrics [552], continuous-time formulation [165], and adaptive thresholds [728] are more some of the more recent ICP advances. Other methods [636, 393] opted to evaluate differences in a probability distribution of a local neighborhood than using Euclidean distances.

Another well-known distribution-based matching is the NDT [58]. NDT divides an input point cloud into a set of voxels and fits a normal distribution to the points in each voxel (see Chapter 6.3.2.2 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.

#### 10.2.1.2 Determining Correspondences

The second core component 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  $p \in P$  in the target  $Q$ , where we use the iteratively updated transformation  $(\mathbf{R}^{k-1}, \mathbf{t}^{k-1})$ , which

is given by:

$$C = \{(\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in P, \mathbf{q} = \arg \min_{\mathbf{q}' \in Q} \|\mathbf{p} - (\mathbf{R}^{k-1}\mathbf{q}' + \mathbf{t}^{k-1})\|_2\} \quad (10.3)$$

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 distance-based neighbor search to more quickly find potential candidates.

Several ICP variants used in popular LiDAR odometry systems [817, 552, 643] employ the first strategy to reduce the set of potential correspondences by building maps with reduced candidate sets,  $P' \subset Q$  and  $Q' \subset Q$  by determining points that fulfill certain geometric criteria. The criteria used include determining points lying on edges or surfaces [817, 552], removing less descriptive points that correspond to the ground plane [642], or downsampling of the target scan [728, 165]. 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 [50] or leveraging voxel grids for approximate neighbor search [817, 165, 728]. 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 [552] 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 scan registration system which can align a sequence of scan observations so as to estimate the relative motion of a robot.

### 10.2.2 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,

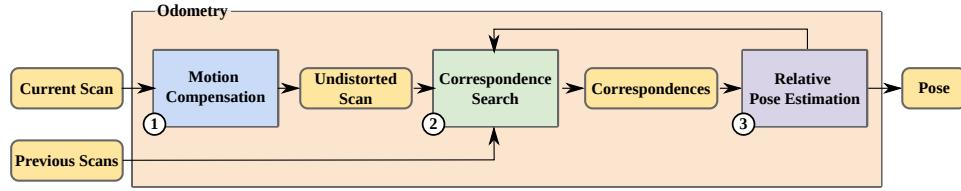


Figure 10.4 Components of a LiDAR odometry pipeline: Given a current scan, (1) motion compensation accounts for the motion of the sensor during the scan process resulting in a undistorted scan. Then, (2) correspondences between the undistorted scan and the previous scans, either a single scan or aggregated 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.

these modules ensure accurate and reliable LiDAR odometry. Figure 10.4 shows the interplay between the different components in a common LiDAR odometry pipeline.

#### 10.2.2.1 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 cloud if left uncorrected<sup>2</sup>. Because of this, 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 [165, 728], however, the constant velocity assumption is inherently less accurate when the motion contains high frequency motions.

<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.

<sup>2</sup> A similar type of distortion effect during camera image capture and is known as the rolling shutter effect.

**Continuous-time Trajectory Optimization** Another widely used approach for motion compensation is continuous-time trajectory optimization techniques using splines [183, 472] and the GP [38]. 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 [472].

**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 [817, 641]. 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 underperform the other simpler methods.

**Point-wise Registration** Being firstly proposed in Point-LIO [298], 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.

#### 10.2.2.2 Feature-based LiDAR Odometry

Once motion distortion is corrected, point correspondences can be established. Similar to visual SLAM (see Chapter 9), leveraging features for scan association is well studied. A feature-based approach allows efficient representation of the scan by 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 [817] was a significant breakthrough in LiDAR SLAM which 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 [51, 495]. 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 [125].
- **Surfels Features** Surfels are small disk-like representations of the surface geometry of a point cloud (see Chapter 6.3.2.2). 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 [556, 50, 587].
- **Intensity Features** Intensity features [179, 278] 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.

#### *10.2.2.3 Direct Point-wise LiDAR Odometry*

A problem with this feature-based approach, 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 scales linearly with the LiDAR scan. The efficiency of this approach can become eroded 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. [835] 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. [774] 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.

#### 10.2.2.4 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 10.2.1).

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 [308], 3D LiDAR odometry [774, 165, 728], and 3D LiDAR SLAM systems [552, 50, 556] 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 [817] 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 [165,

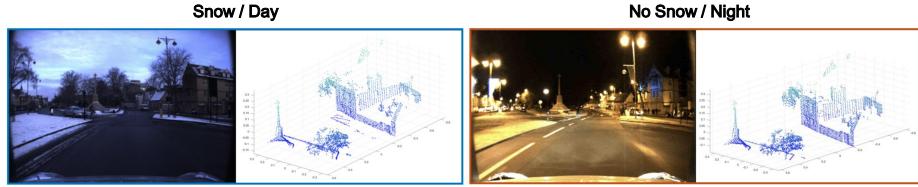


Figure 10.5 A key attribute of LiDAR is that the measurements of an environment are not affected by lighting conditions. This is a key advantage for LiDAR place recognition versus image-based place recognition. These images show camera and LiDAR data of the same place from different weather/lighting conditions (courtesy of Uy and Lee [721]). While the visual appearance changes drastically, the 3D measurements produced by the LiDAR sensor are very similar.

728] 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 [436], and this line of work has been extended to unsupervised learning methods [138]. While the performance of deep LiDAR methods are generally promising, concerns have been raised regarding their generalization capabilities.

### 10.2.3 Summary

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.

## 10.3 LiDAR Place Recognition

Place recognition systems seek to identify places that have been previously visited by a robot/sensor. It is an essential capability for several applications, 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 cameras as shown in Figure 10.5. 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 (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 [186, 821] or compute global descriptions [378, 776] (*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 [104]. Seminal papers and paradigms in the area of LiDAR-based place recognition will be revisited in more detail in Section 10.3.2.

**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. Global LiDAR descriptors such as ScanContext [378] typically fail in such situations. Other approaches using object-level clusters, such as SegMap [186] and InstaLoc [821], 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.

### 10.3.1 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 composed 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 10.4.1). Secondly, working towards the more complex goal of awareness can naturally enhance the invariance capability (*e.g.*, estimating heading changes [373] or inferring the degree of overlap [126]).

### **10.3.2 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.

#### *10.3.2.1 Local Descriptors*

In the early days of LiDAR place recognition research, and corresponding to the evolution of visual place recognition methods (*e.g.*, SIFT [461], ORB [617], DBoW2 [246]), computing local keypoint descriptors was a natural approach both for 2D [701] and 3D LiDAR sensors [69]. However, 3D local descriptors specifically developed for dense RGB-D point cloud registration or object recognition [620, 621, 706] 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) [312]. However, these descriptor remain limited to a local neighborhood,

which results in poor descriptiveness due to the lack of metric structural context from across an overall scan.

#### 10.3.2.2 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++ [373, 378] and RING++ [465, 776] 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 6.1) provides a structured, well-aligned representation of a scan. The recent advancements of dense multi-beam LiDAR technology (see Section 10.1), 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) [404] or Vision Transformers [181] to extract a detailed feature representation. OverlapNet [126, 477] showed that yaw invariance can be achieved by applying an overlap loss to the range image, while more recently FRAME [661] demonstrated LiDAR place recognition in mines using range images.

#### 10.3.2.3 High-level or Combined Descriptors

To derive a descriptor for a scan (either local or global), there have been also approaches proposed that 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 [186] and InstaLoc [821].

<sup>3</sup> As an example RING++ [776] uses a representation of 120 by 120 pixels equally spaced over a  $[-80, 80]$  meter grid.

**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 predictable directions along roads. To address this issue, BTC [812, 813] 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 [721, 104] 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. [104] 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.

#### 10.3.3 Summary

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 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.

## 10.4 LiDAR SLAM

The purpose of the LiDAR odometry systems described in Section 10.2 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.

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.

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 our current observations and past observations a single map representation. This task is particularly challenging in large-scale and potentially dynamic environments.

In Section 10.2, 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 10.3 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 [50, 556, 182, 171, 810] are composed of the components shown in the system diagram in Figure 10.6. In the following Section 10.4.1, we will discuss this structure in more detail. We will then focus on the key steps of backend optimization and map update in Section 10.4.2 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 10.4.3, 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 10.5.

#### **10.4.1 Structure of a LiDAR SLAM System**

When implementing a LiDAR SLAM system [50, 556, 182, 171, 810] 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.*, 10Hz) with the other modules operating at a lower rate (*e.g.*, 1 Hz.).

More concretely, consider Figure 10.6. Here, (1) an odometry component esti-

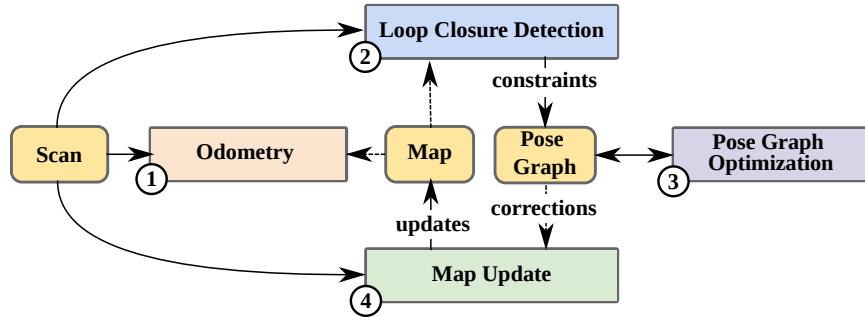


Figure 10.6 Conceptual structure of a typical LiDAR SLAM system composed of multiple components: (1) Odometry estimates the robot/sensor pose, (2) Loop Closure Detection determines if a place has been revisited, (3) Pose-graph optimization uses the loop closure constraints to correct the pose trajectory using factor graph optimization, and (4) Map Update uses the most recent pose trajectory to revise the map representation.

mates 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 10.2). 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 10.3. 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 [786, 441]. 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.

Module (3) is the backend pose-graph optimization (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

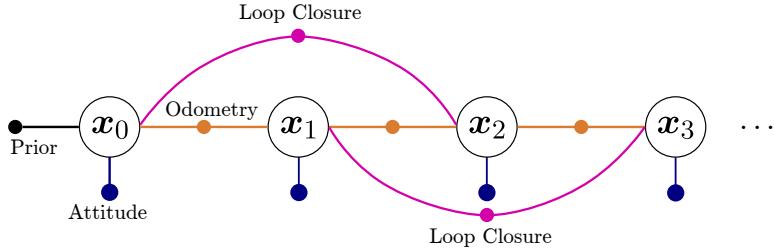


Figure 10.7 A SLAM problem represented as a pose-graph. Each node represents the pose of the sensor whereas the edges represent the constraints coming from odometry (orange) and loop closures (magenta). A Prior Factor fixes the graph origin. Optional Attitude Factors can be used to constrain the pitch and roll when inertial sensing is available.

points directly [165] while approaches such as surfels [182, 556, 50] and implicit representations [810, 171] attempt to improve the quality of the underlying map or seek to achieve a stronger probabilistic foundation. We refer the reader to Chapter 6 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.

#### 10.4.2 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 is sometimes known as the *SLAM backend* is the module which corrects the estimated trajectory to respect both the odometry constraints and loop closure constraints due to revisited places. As introduced and deeply discussed in Part I, see Chapter 2, a factor graph can be used to represent these constraints (as shown in Figure 10.7). Because the graph is made up of only pose constraints, it is commonly referred to as a pose-graph. The changes in the map An example of a point cloud map before and after loop closure detection and pose-graph optimization is shown in Figure 10.8.

The constraint set can be optimized using general-purpose solvers such as g2o [408] and GTSAM [162]. 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 please see iSAM2 [354] and HOG-Man [270].

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 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 [411, 107] — 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 [16, 102, 682, 683, 785] which can down-weight, disable or ignore pose-graph constraints which would otherwise cause the map to degrade or diverge.

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 [556] or to directly link map elements (*i.e.*, such as surfels or submaps) to poses, to allow map deformation in a more scalable manner.

#### **10.4.3 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 10.9.

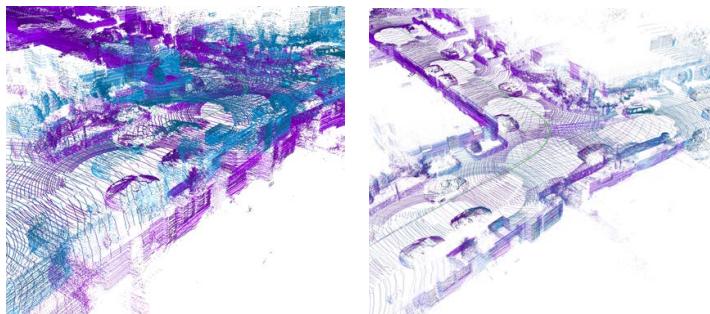


Figure 10.8 By incorporating loop closure constraints, a SLAM system can create a globally consistent map of revisited locations. The left shows the odometry-only trajectory of a revisited place with visible misalignment between the original visit (blue) and the current visit (purple) to this junction. The right shows the result after pose-graph optimization when loop closures have been integrated resulting in a consistent map of the road junction.

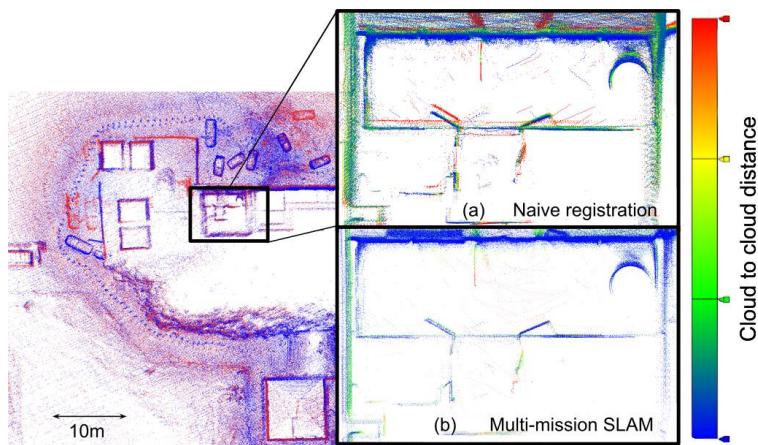


Figure 10.9 Comparison between (a) naive direct alignment of two global point clouds and (b) multi-mission pose-graph relaxation. (a) Point-to-point distances between the two global point clouds shows “double walling” causing phantom change to be hallucinated. (b) Multi-mission relaxation reduces point-to-point distances with structures being more clearly reconstructed. From [615].

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. [372] introduced an auxiliary variable called an *anchor node* which accounts for the

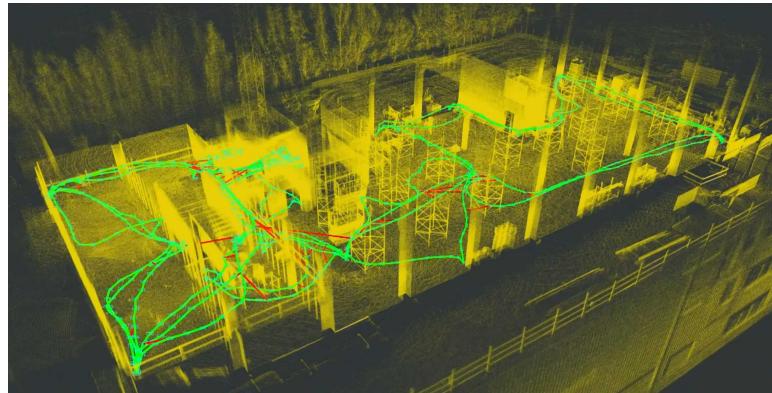


Figure 10.10 A multi-session SLAM map of a construction site. Five different mapping sessions are merged together by establishing inter-session loop closure constraints (in red) and adjusting a joint optimization of the five mapping session trajectories (in green).

different map coordinates of the individual SLAM missions. This node allows easy global alignment of each individual map and has been used for both multi-session visual SLAM and LiDAR SLAM [491, 263].

As described in [185], 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.

#### *10.4.3.1 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



Figure 10.11 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.

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.

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 [548] 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 [399] 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, kerbs 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 [197]. 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 [400] 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 [329, 700] 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.

## 10.5 Outlook and Futures Challenges

LiDAR SLAM has seen significant advancements over the last decades — especially since the introduction of LOAM [817]. Improved odometry with high accuracy [774, 643, 728, 165] and efficient pose-graph SLAM systems [50, 171, 556, 587] 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.* [831] 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 [197].

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 [728]. 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 [651, 806] offer a comprehensive foundation on the topic. Ongoing research includes methods for robust retrieval which generalize across the various categories of LiDAR sensors [351]. Other research looks to achieve heterogeneous place recognition between LiDAR and other modalities such as radar [804] and OpenStreetMap [139]. Researchers have also successfully leveraged the LiDAR’s intensity information [644, 735], alongside traditional XYZ data to improve performance. Long-term place recognition across multiple mapping sessions [375] is another promising research topic; as is change detection and lifelong map management [263, 808].

**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 [761, 829]. 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 repeatedly. 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 cannot provide a calibrated and robust estimate of the pose uncertainty.

While some early approaches [411, 107] 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 Autonomy 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 parameters of Size, Weight and Power (SWaP) are augmented with additional parameters of accuracy, robustness, computation and latency.

The ability for a self-driving car to respond to a potential upcoming collision with as little delay soon as possible is affected by the computational latency of the

LiDAR system. Thus in practical application the most accurate and computationally complex system may not always be the preferred solution.

# 11

## Radar SLAM

Martin Magnusson, Christoffer Heckman, Henrik Andreasson, Ayoung Kim,  
Timothy Barfoot, Michael Kaess, and Paul Newman

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 weather, 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. We then discuss radar filtering, radar odometry, place recognition, and finally SLAM; Figure 11.1 shows how these pieces fit together. We conclude with an outlook on the use of radar moving forwards.

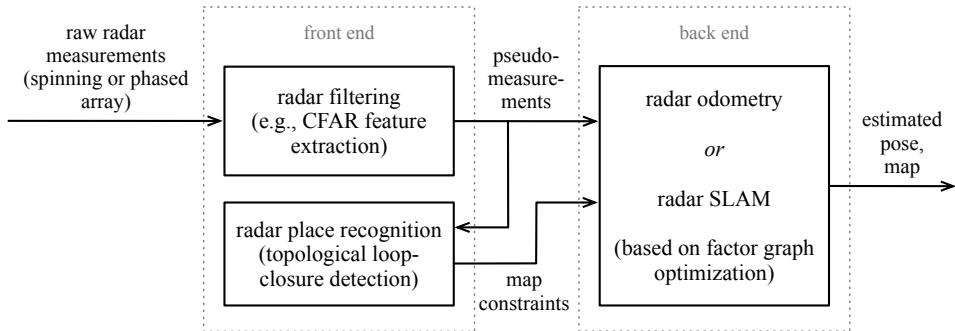


Figure 11.1 The flow of information in radar-based SLAM follows the same pattern as other SLAM systems with the details of each process being slightly altered to suit the specifics of radar.

### 11.1 Introduction to Radar

#### 11.1.1 Sensor Types

We detail in the following section two of the most common radar types that are encountered in robotics: *spinning radar* and *system-on-a-chip (SoC) radar* (see Figure

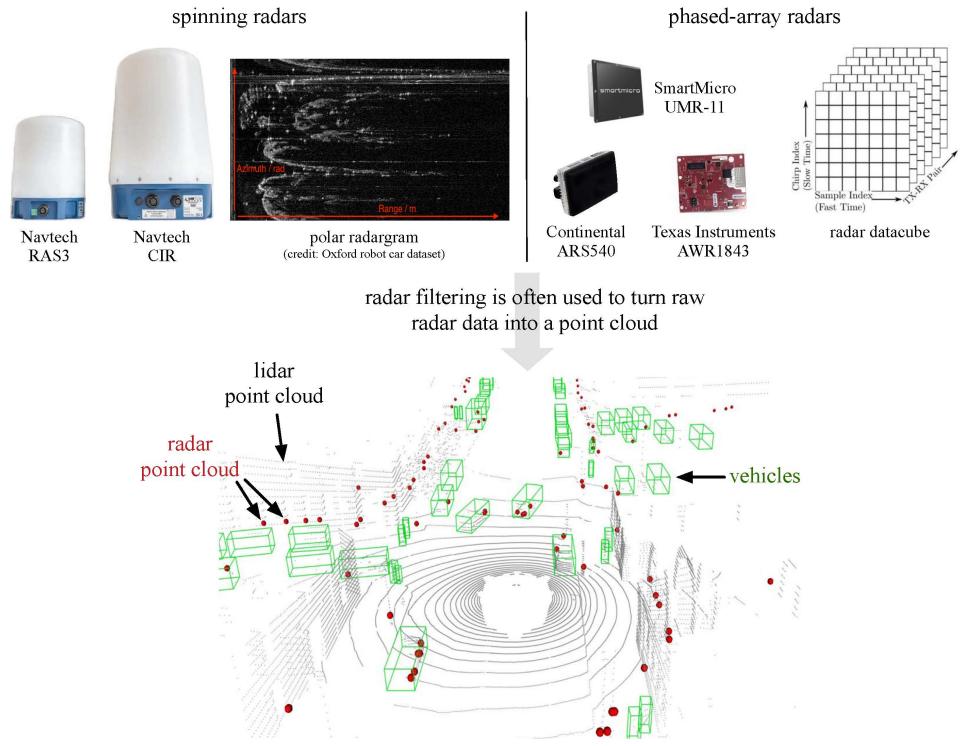


Figure 11.2 The two main categories of radars used in robotics are *spinning* and *phased-array SoC*. Some examples of each are shown along with the main data product produced by each type of sensor. Often these raw data products are subsequently turned into a sparse point cloud using radar filtering (point cloud image source: [524]). The point cloud density can vary quite a bit based on the type of radar and filtering being used. For example, point cloud can be 2D for imaging radars and 3D for phased-array radars.

11.2). 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.

#### 11.1.1.1 Spinning Radar

Utilizing a rotating radar sensor, *spinning radar* – sometimes referred to as scanning radar or imaging radar – crafts precise polar representations of its surroundings, exemplified in Figure 11.2. 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 wave path. Additionally, these radars sometimes do not offer velocity data due to their operational mechanism, which involves transmitting and receiving a single pulse for each antenna angle, unlike the multiple pulses necessary for calculating velocity. More recent work uses the signal from multiple neighbouring azimuths to recover velocity.

#### 11.1.1.2 SoC Radar

*System-on-a-chip* 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 main data product 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.

#### 11.1.2 Radar Sensing Principles

In this section, we will discuss basic operations, antenna 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.

##### 11.1.2.1 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), 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 an imaginary 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 the result of multiplying the radar's emitted power by 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.

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.

#### 11.1.2.2 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* later on. These chirps bounce off objects in the environment, and the RX antenna captures the echoes (see Figure 11.3). Upon receiving a chirp, the signal is amplified and then mixed with, or deducted 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 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 measure-

<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.

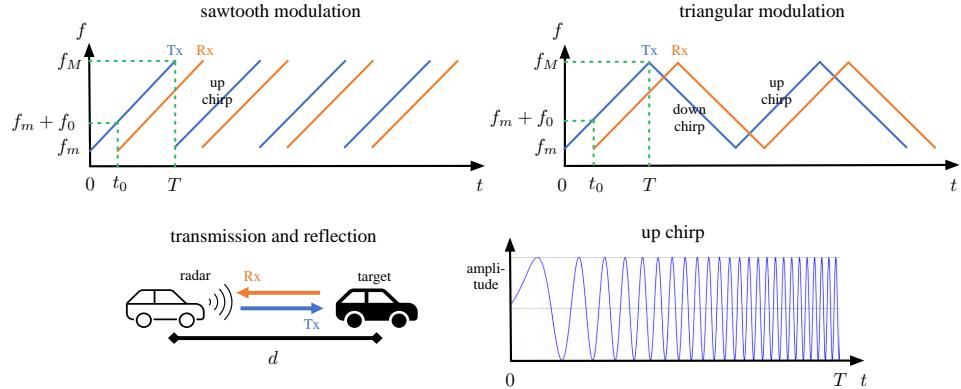


Figure 11.3 A frequency modulated continuous-wave (FMCW) radar works by emitting *chirps* (waves of increasing or decreasing frequency) that are reflected off targets and then received. Range (and velocity) are determined by analyzing the frequency (and phase) shifts of the reflected signal compared 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.

ments 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

$$d = \frac{c}{2} t_0. \quad (11.1)$$

The top-left portion of Figure 11.3 shows the region where an IF signal is generated using green dashed lines. The difference between the transmitted and received 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

$$f_0 = \frac{2Bd}{Tc} = \frac{2Sd}{c} \Rightarrow d = \frac{c f_0}{2S}, \quad (11.2)$$

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 sam-

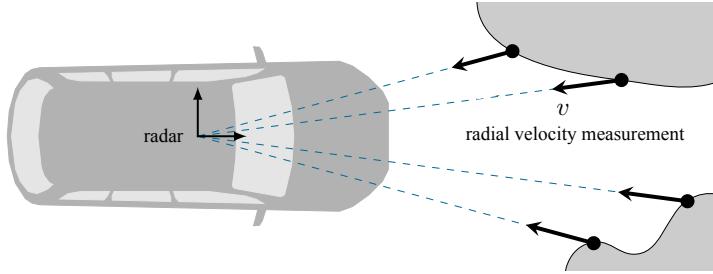


Figure 11.4 Radar is able to use the Doppler effect to measure the *radial* velocity between a unit and the scene it is imaging.

pling. 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.

#### 11.1.2.3 Determining Distance and Velocity with Sawtooth Modulation

Sawtooth frequency modulation (see Figure 11.3) 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  $d$ :

$$\phi_0 = 2\pi f_m t_0 = \frac{4\pi d}{\lambda}. \quad (11.3)$$

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 11.4), at least two sequential up chirps are employed as shown in the top left of Figure 11.3. 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 from the phase difference,

$$\Delta\phi = \frac{4\pi\Delta d}{\lambda} = \frac{4\pi v T}{\lambda}, \quad (11.4)$$

simplified to

$$v = \frac{\lambda \Delta\phi}{4\pi T}. \quad (11.5)$$

As velocity is a function of phase, we note the maximum detectable velocity is unambiguous for  $|\Delta\phi| < \pi$ . Thus  $v_{\max} = \lambda/(4T)$  a function of the wavelength of the signal and the time between chirps.

#### 11.1.2.4 Determining Distance and Velocity with Triangular Modulation

Triangular frequency modulation (see Figure 11.3) 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,  $f_{0,d}$ :

$$f_{0,\text{up}} = f_{0,t} + f_{0,d}. \quad (11.6)$$

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:

$$f_{0,\text{down}} = -f_{0,t} + f_{0,d}. \quad (11.7)$$

From these two equations we can solve for  $f_{0,t}$  and  $f_{0,d}$ :

$$f_{0,t} = \frac{f_{0,\text{up}} - f_{0,\text{down}}}{2}, \quad f_{0,d} = \frac{f_{0,\text{up}} + f_{0,\text{down}}}{2}. \quad (11.8)$$

Finally, from these two components, we can calculate the range and velocity according to

$$d = \frac{cf_{0,t}}{2S}, \quad v = \frac{cf_{0,d}}{2ST}. \quad (11.9)$$

Notably, the range calculation presented earlier in (11.2) 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.

#### 11.1.2.5 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 (11.3) with the geometric relation  $\Delta d = \ell \sin \theta$ . The received signal must travel an

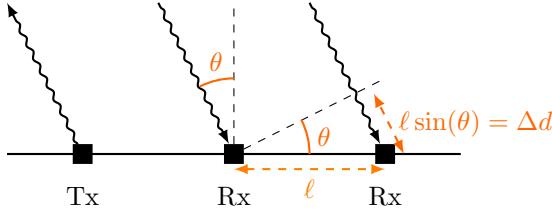


Figure 11.5 Angle of arrival estimation with phased-array radar. The measured difference in phase of the signal emitted by the TX transmitter antenna as received by two RX receiver antennas at a distance  $\ell$  corresponds to the angle  $\theta$  to the target.

extra distance  $\ell \sin \theta$  to reach the second receiver antenna, as illustrated in Figure 11.5. 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

$$\theta = \arcsin \frac{\lambda \Delta\phi}{2\pi\ell}. \quad (11.10)$$

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  $\Delta\phi$ .

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) or code division (where each transmitter sends a signal modulated by a unique code sequence). 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.

### 11.1.3 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 peculiar to radar, such as spurious signals across the sensor’s range,

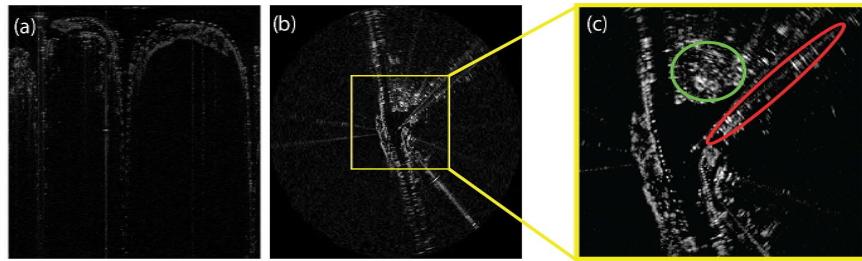


Figure 11.6 A polar radargram in (a) transformed to Euclidean coordinates in (b), where we can observe several types of radar noise that are unique compared to other sensors from a zoomed view in (c). Speckle noise returns are the most common, with ambiguous clutter circled in green. Multipath reflections 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 [376].

complex speckle noise, and multi-path reflections, which result in multiple signals being returned from a single object. Illustrations of some of these noise phenomena are provided in Figure 11.6, which depicts a polar radargram transformed to Euclidean coordinates. We discuss some relevant radar filter techniques in Section 11.1.4.

#### 11.1.3.1 Speckle Noise

Radar operates in environments that introduce various types of noise, including thermal noise, electronic flaws, and fluctuations in the RCS of targets. When a radar emits an electromagnetic pulse, it captures the energy reflected back by objects within its surroundings. The interaction of this pulse with objects scatters the radar waves, leading to constructive and destructive interferences. 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.

#### 11.1.3.2 Multipath

Beyond speckle noise, multipath interference constitutes another form of erroneous return, 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.

#### *11.1.3.3 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.

#### *11.1.4 Radar Filtering*

The occurrence and distribution of false targets (Section 11.1.3) 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.

The constant false alarm rate (CFAR) filter [534] 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. Figure 11.7 illustrates three CFAR

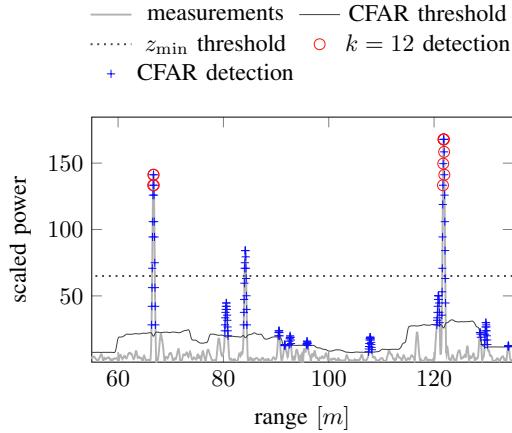


Figure 11.7 Examples of radar filtering, comparing a CFAR filter with a constant power threshold and a  $k$ -strongest strategy. The power/range plot along one azimuth direction is plotted in grey. Returns from true targets appear as spikes in the plot but there are also several ambiguous peaks. CFAR 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. [10].)

approaches, with cell-averaging CFAR recognizing one return, and both ordered statistic and censored CFAR identifying two returns. This method ensures local noise factors are considered while still recognizing high-intensity returns, allowing for variation in the threshold across the frequency spectrum.

A variant of CFAR designed and tested specifically for radar odometry is BFAR (bounded false alarm rate) [23] 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. 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. [129] use a generative adversarial network (GAN) to generate point clouds based on range-Doppler velocity matrices. Xu et al. [773] 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.

## 11.2 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 11.2.1. In addition to Doppler-based methods, the relative 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 11.2.2, to extract higher-level features containing information from the raw signal (see Section 11.2.3) or to extract range points that can be used in registration, much like in LiDAR odometry (see Section 11.2.4).

An indicative example of what open-loop radar odometry using a 2D scanning radar may look like is shown in Figure 11.8.

### 11.2.1 Doppler Odometry

The radial velocity obtained from Doppler measurements can be used to directly estimate the sensor’s linear velocity. In Doer and Trommer [177], 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, knowing where the radar is mounted with respect to the center of rotation and assuming no skidding [365, 239]. Therefore it is common to use IMU data (or more specifically, a gyroscope) for radar odometry systems that relies solely on Doppler information from a single radar. In Huang et al. [328], a consumer grade IMU combined with cascaded SoC radars achieves low drift in diverse 3D indoor spaces.

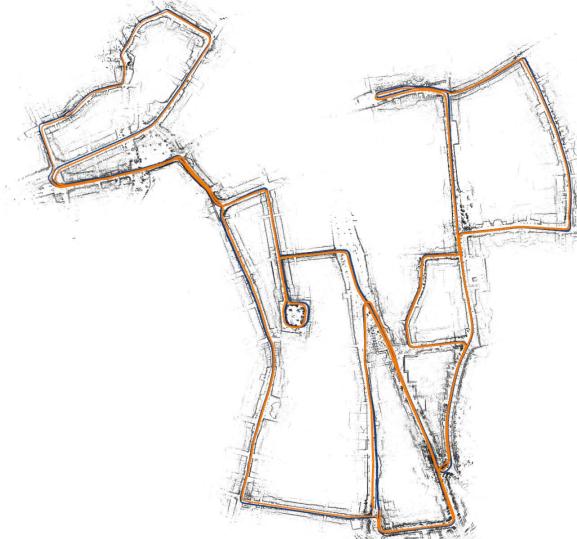


Figure 11.8 Example of open-loop radar odometry on the Oxford Radar Robotcar dataset [82], using the CFEAR method [12] with data from a Navtech 2D spinning radar. The ground-truth trajectory plotted in blue and the odometry estimate in orange. Point targets extracted from the radargram in grey. This figure is adapted from [12].

Kubelka et al. [405] compared several variants of registration-based approaches for 3+1D radar with registration-free Doppler + IMU odometry [177] 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. [365] 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. [239] 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. [402].

#### *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

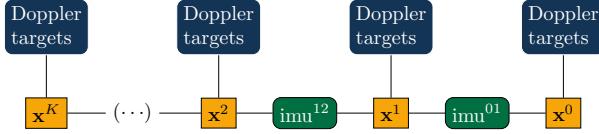


Figure 11.9 Factor graph representation of the radar-inertial velocity estimation system. States from  $K$  previous timesteps are jointly estimated using Doppler targets and sets of IMU measurements as constraints. Figure adapted from Kramer et al. [402].

using a factor graph, where states from  $N$  previous time steps are jointly estimated using Doppler targets and sets of IMU measurements as constraints.

Accelerometer measurements are typically affected by both bias  $\mathbf{b}_a$  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  $\mathbf{q}_{WS}$ . To accurately estimate the IMU's attitude, gyro measurements are used, necessitating the estimation of gyro biases  $\mathbf{b}_g$ . Consequently, the full state vector is expressed as  $\mathbf{x} = [\mathbf{v}_S^T, \mathbf{q}_{WS}^T, \mathbf{b}_g^T, \mathbf{b}_a^T]^T$ .

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

$$J(\mathbf{x}) = \underbrace{\sum_{k=1}^K \sum_{d \in \mathcal{D}^k} e_d w_d}_{\text{Doppler term}} + \underbrace{\sum_{k=1}^{K-1} \mathbf{e}_s^{k^T} W_s^k \mathbf{e}_s^k}_{\text{inertial term}} \quad (11.11)$$

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  $k$ ,  $e_d$  is the Doppler velocity error, and  $\mathbf{e}_s$  is the IMU error. The error terms are weighted by the information matrix  $W_s$  in the case of the IMU errors; and the normalized intensity of the corresponding radar target

$$w_d^j = \frac{i_j}{\sum_{d \in \mathcal{D}} i_d} \quad (11.12)$$

in the case of the Doppler velocity measurements where  $w_d^j$  is the weight for target  $j$  in scan  $\mathcal{D}$  and  $i_j$  is the intensity of target  $j$ . In the following sections we detail the formulation of our Doppler and IMU measurement constraints.

A radar measurement consists of a set of targets  $\mathcal{D}$ . Each  $d \in \mathcal{D}$  consists of  $[r_S, v_R, \theta_S, \phi_S]^T$  representing the range, Doppler (radial) velocity, azimuth, and elevation for target  $d$ . The Doppler velocity measurement  $v_R$  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  $\mathbf{r}_S$ . 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

$$\tilde{v}_R - e_v = \mathbf{v}_S \left( \frac{\tilde{\mathbf{r}}_S}{\|\tilde{\mathbf{r}}_S\|} \right)^T. \quad (11.13)$$

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:

$$e^k(\mathbf{x}^k, \mathbf{d}^{i,k}) = v_R^{i,k} - \mathbf{v}_S^k \left( \frac{\mathbf{r}_S^{i,k}}{\|\mathbf{r}_S^{i,k}\|} \right)^T \quad (11.14)$$

where  $x^k$  is the state at time  $k$  and  $\mathbf{d}^{i,k}$  is the  $i^{\text{th}}$  target in the set of radar measurements 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.

### 11.2.2 Direct Odometry

Methods that operate on raw radargrams as the ones depicted in Figure 11.6 and Figure 11.2 are denoted as *direct*.

Direct approaches make use of classical signal processing techniques as phase correlation and the Fourier-Mellin transform [117, 557]. Given two sequential polar radargrams, as in Figure 11.2, 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 similarly computing the correlation between the images in a Cartesian frame (Figure 11.6). 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. [41]. 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.

### **11.2.3 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 [94, 432], SURF [317], and ORB [361] features. (Callmer et al. [94] match large-scale features of islands in an archipelago and Li et al. [432] 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 [318]. Feature descriptors may also be place-variant in radar compared to other sensors, making it difficult to do data association if the sensor pose is different. (See also Section 11.3.) FSCD and BASD [589, 635] 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 [40]. One drawback is that training data including radargrams along with ground truth poses from a somewhat similar environment are required.

### **11.2.4 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 11.6 we first need to select which points to use by filtering those signal returns that do not correspond to a relevant peak. (These filtering techniques can be seen as similar to the key-point extraction in Section 11.2.3 but without extracting descriptors.) As discussed in Section 11.1.4, many approaches extract a set of range readings per azimuth; *e.g.*, using CFAR (Figure 11.7, Section 11.1.4), using noise statistics to remove redundant or noisy readings [106], BFAR [23], or simply the  $k$  strongest returns per azimuth. An exception is Kellner et al. [364], 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 [129, 773]. A key challenge is to extract an adequate amount of readings; too few readings discards relevant information and too many include noise [106]. For example, CFAR has been found difficult to tune in this respect [82].

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 10.2), 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 [82], power-shifted NDT [409], and CFEAR [12] (not to be confused with the CFAR method for filtering which is discussed in Section 11.1.4) – 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. [409] 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. [82] extract point clouds from radargrams using BFAR [23] 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 11.2.1) as an initial estimate and perform point cloud registration to refine it. The registration-based odometry component in 4DRadarSLAM [818] uses a variant of GICP [636] that is adapted for 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 [845] employ 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 [845, 818] are further described in Section 11.4.2.) The EFEAR-4D method [767] extends CFEAR 2D odometry [12] 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.

### **11.2.5 Motion Compensation**

As discussed in Section 11.1.3.3, 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 [12]. 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

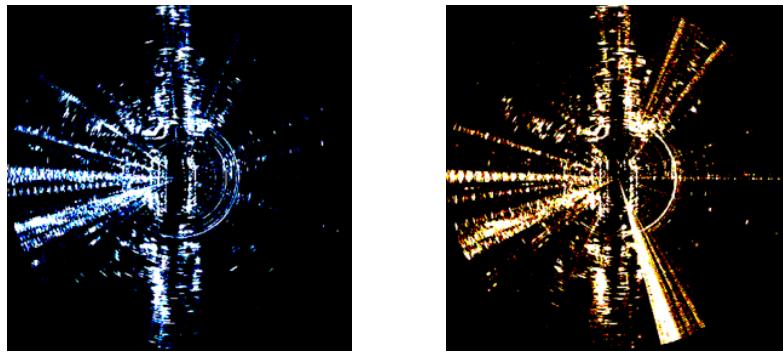


Figure 11.10 Two radar scans obtained from the same location. The noise pattern induces visual aliasing, complicating the process of place recognition.

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. [529] a spline representation is implemented in a pipeline that uses automotive SoC radars. Gaussian processes are used in Burnett et al. [84] to form a factor graph representing the trajectory by combining an IMU sensor with a spinning radar.

### 11.3 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 10. Radar PR can be categorized as either appearance-only topological PR or pose-based metric PR. As discussed in the chapters on visual and LiDAR PR, securing invariance to 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.

#### 11.3.1 Unique Challenges in Radar Place Recognition

While, in principle, several of the methods 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 11.1.3).

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 11.6. 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 11.10. 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 [376] 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.

### ***11.3.2 Learning-based Radar PR***

For spinning radars, treating the 2D radargram as an image, 2D image retrieval has been leveraged for place recognition.<sup>2</sup>

Saftecu et al. [622] 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.

<sup>2</sup> A more comprehensive overview can be found in [79].

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. [158] 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 [237].

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 size, require slightly different treatment since they typically have lower range and smaller field of view. Cai et al. [93] 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 [32] layer after which a re-ranking 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. [304] 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. [93], Herraez et al. [304] 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. Using Transformer in radar for the first time, Peng et al. [562] filters noise points in a similar way as the Doppler odometry methods in Section 11.2.1. 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, Peng et al. [562] demonstrate a radar-specific feature extraction backbone named MinkLoc4D which takes inspiration from LiDAR place recognition architectures [851].

### **11.3.3 Descriptor-based Radar PR**

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 [379], RING [776], or M2DP [299]; 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. [318] implement place recognition by adapting the M2DP descriptor [299] 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. [340] modify the RING descriptor [776] 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. [13] 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 11.3.1. 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 [818, 845, 438]. 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 [736] 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°) [818].

## 11.4 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 11.4.3.

### 11.4.1 Map Representations

Many radar SLAM systems generate maps that rely on similar representations discussed in Chapter 6. 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 [146, 176, 94, 635]. This target detection can also be used for occupancy grid map as in the series of works by Mullane et al. [512], while exploiting the detection probability [513] 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 [483, 498] or point cloud maps [318]. The extracted point cloud can also be augmented with additional information, as in [13] 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.*, 48×48 antennas for the Sensrad Hugin radar) [818, 845].

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 [614]. 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 [401], 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 [322], 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. [538] 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 [773, 28], where LiDAR data serves as ground truth to achieve higher resolution and reduced noise in radar outputs. A.N. Mopidevi [28] builds 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 [501] and later applied for LiDAR data [834, 679] have been applied to 2D radar data as well [66]. 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. [66], 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.

#### **11.4.2 Radar SLAM**

In the preceding sections we have covered the main components that make up graph-based radar SLAM frameworks: open-loop odometry estimation, place recognition



Figure 11.11 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.

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 radar SLAM pipelines with proprioceptive sensors (*e.g.*, wheel odometry and IMU) and discusses how they implement the components. These pipelines generally follow the same architecture as shown in Figure 11.1, 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 [13] builds upon the CFEAR 2D radar odometry method discussed in Section 11.2. The pose of the sensor is tracked using every radar scan in 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 [13] 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 [374] (see 11.3.3) 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 [11] 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. [318] 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 [466]. 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 [13]; 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 [299] 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. [635]. 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 [589]). 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 who 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 [818] and 4D iRIOM [845], 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 [636] 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 [736] in order to avoid uninformative descriptors due to noisy elevation measurements. 4DRadarSLAM additionally in-

cludes 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.

#### **11.4.3 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. [231, 232] 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 [803, 805].

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. [319] demonstrate how 2D scanning radar data can be used to localize in prior public maps such as OpenStreetMap. Their system runs odometry using RadarSLAM [318] 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 [687], 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 capabilities 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 response 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 [631] and Takeuchi et al. [684] 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. [579] 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.

Doer and Trommer [178] extend ROVIO [64], which is a filter-based visual-inertial odometry approach, to integrate radar egovelociy estimates using the Doppler odometry approach described in Section 11.2.1 [177]. In a similar way, also thermal camera data can be fused with radar to achieve a multi-modal radar-thermal estimation pipeline [178]. Zhang et al. [819] 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.

## 11.5 Radar Datasets

In this section, we briefly summarize notable datasets from the radar SLAM literature. The datasets are also listed in Table 11.1.

*Spinning Radar* The datasets with spinning radar in Table 11.1 all use Navtech 2D radar sensors. Two of the first large-scale radar datasets for odometry and SLAM are the Oxford Radar Robotcar dataset [42] and MuRan [377]. 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 MuRan 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. MuRan focuses on facilitating PR research but has also been well used to benchmark odometry and SLAM methods. The Boreas dataset [83], 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 [238] is focusing on non-urban driving, in contrast to the other datasets in Table 11.1. 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 [92] 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 [634] is a dataset with 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 [403] 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’ (radargrams) 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 [820] and MSC-RAD4R [140] 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 [324] 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.

## 11.6 Outlook and challenges

Radar SLAM pipelines that work in 3D are still rather few but as high-resolution SoC sensors develop we can expect there to be more work on fully 3D odometry and place recognition for radar. This will be particularly important on drones, for example, where radar is less utilized today. However, 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. Creative designs will be required to enable 3D wide field-of-view radar units. In the meantime, we are likely to see significant advancements in handling constellations of several small

	Dataset	Lidar	Cameras	Ground truth	Environment	Inclement Weather
<b>Spinning radar</b>	Oxford Radar RobotCar [42]	Yes	Stereo/Mono	GPS/IMU + VO	Dense Urban	Rain, Fog
	Boreas [83]	Yes	Mono	GPS/IMU + RTK	Sparse Urban	Rain, Snow, Fog
	MulRan [377]	Yes	No	SLAM	Mixed Urban	–
	RADIATE [645]	Yes	Stereo	GPS/IMU	Mixed Urban	Rain, Snow
	OORD [238]	Yes	Mono	GPS	Urban and Offroad	Snow, Night
<b>SoC array radar</b>	nuScenes [92]	Yes	Stereo	GPS/IMU	Mixed Urban and Natural	Rain
	RadarScenes [634]	No	Mono	None	Mixed Urban Roadways	Rain, Fog
	ColoRadar [403]	Yes	No	SLAM	Varying	–
	NTU4DRadLM [820]	Yes	Mono	SLAM	Mixed Urban	–
	MSC-RAD4R [140]	Yes	Stereo	GPS + RTK	Mixed Urban	Smoke, Snow, Night
	Snail [324]	Yes	Stereo	TLS	Roadways and Tunnels	Rain, Night
	K-Radar [550]	Yes	Stereo	GPS/IMU + RTK	Roadways	Rain, Snow, Fog, Night
	TruckScene [218]	Yes	Stereo	GPS/IMU + RTK	Roadways	Rain, Snow, Fog, Night
<b>Both</b>	HeRCULES [380]	Yes	Stereo	GPS/IMU + RTK	Mixed Urban and Natural	Rain, Snow, Night

Table 11.1 *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.*

field-of-view (FoV) radars in SLAM. This naturally comes along with calibration and other issues.

There will also likely be a shift towards making better use of the raw radargrams and datacubes, which contains 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.

# 12

## Event-based SLAM

Guillermo Gallego, Javier Hidalgo-Carrió, and Davide Scaramuzza

An inquisitive reader would notice that SLAM is paramount in applications that involve interpretation of spatial relationships and interaction with the surroundings to solve complex real-world problems. SLAM’s primary sensors are critical for the system’s success and adaptability. Visual SLAM is the most extended category of all because cameras are broadly available (affordable) and produce an intuitive and informative signal that allows us 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 intelligence 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. 12.1.

### 12.1 Sensor Description

#### 12.1.1 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 Sensor (DVS) [440, 245] operate independently from each other, responding to brightness changes in the scene asynchronously, as they occur (Figure 12.2b). 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 12.2a).

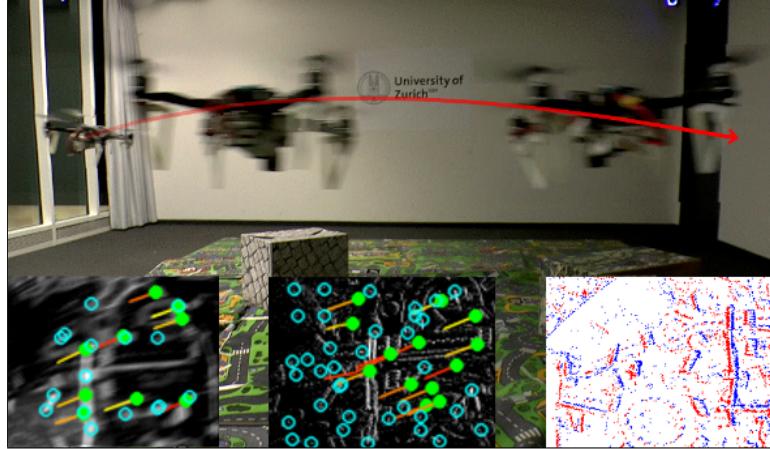


Figure 12.1 Drone with a downlooking DAVIS camera [76] ( $240 \times 180$  px) performing an autonomous flight using a visual-inertial odometry (VIO) algorithm [609] for state estimation. The high speed and high dynamic range of the event camera data are leveraged to operate in difficult illumination 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 [609] is also demonstrated on high-speed scenarios, such as an event camera spinning tied to a rope. Image from [245].

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 12.2). When the change reaches a threshold  $C$ ,

$$\Delta L \doteq L(\mathbf{x}_k, t_k) - L(\mathbf{x}_k, t_k - \Delta t_k) = p_k C, \quad (12.1)$$

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 per second are produced because each pixel adapts its sampling rate to the rate of change of the intensity signal that it monitors.

*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

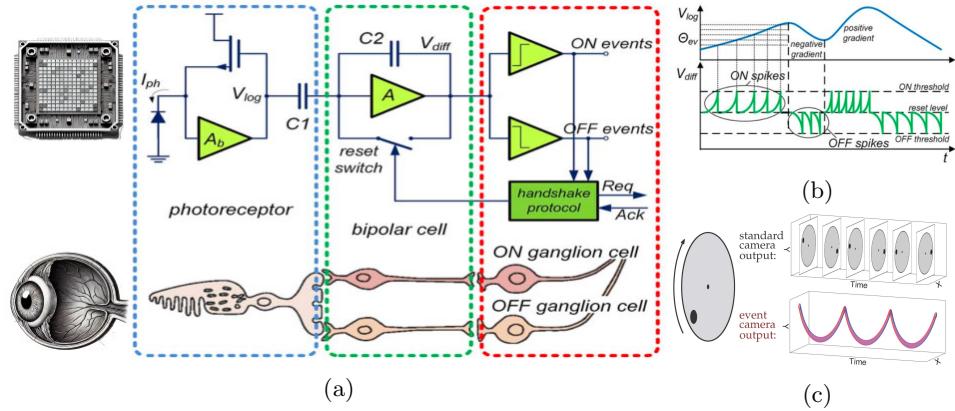


Figure 12.2 Working principle of an event camera (e.g., DVS): (a) Three-layer model of a human retina and corresponding DVS pixel circuit; (b) Schematic of the operation of a DVS pixel, converting light into events (spikes), with the colors of the signals matching those of the layers in (a); (c) Comparison 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). Image adapted from [577].

recognition. The DVS [440] 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 12.2). The three layers realize the functions of light conversion, delta-modulation and comparison, respectively. Cameras like the Asynchronous time-based image sensor (ATIS) [576] or the Dynamic and Active-Pixel Vision Sensor (DAVIS) [76] 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 [577, 245].

### 12.1.2 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 [27].

*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.

#### 12.1.3 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 [245, Tab. 1], [110, Tabs. 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 [440]) to less than 5 µm [219]. 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 [219].

*Grayscale output:* early devices such as the DAVIS or ATIS concurrently output grayscale data (e.g., images [76]), 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 [219] 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) [503].

*Inertial data:* some cameras also provide data from an inertial measurement unit (IMU) integrated in the same device. IMUs are valuable complementary proprio-

ceptive sensors to cameras, enabling visual-inertial odometry (VIO) (sensor fusion), 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 [797], with lower spatial resolution for events than for intensity output, or foveated sensors [214], 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.

## 12.2 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. 12.3). 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 [149, 381] (rotational-motion SLAM) and [382] (6-DoF SLAM). Event-based SLAM and optical flow estimation have been treated together in [149, 799, 653, 391].

### 12.3 Methodology Overview

Event-based SLAM methods can be broadly categorized in two, 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 (12.1) (or its linearized version [245]) 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, 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 with 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. 12.2, it has connections with other problems (optical flow, tracking, segmentation, etc.), in stronger or weaker form depending on the

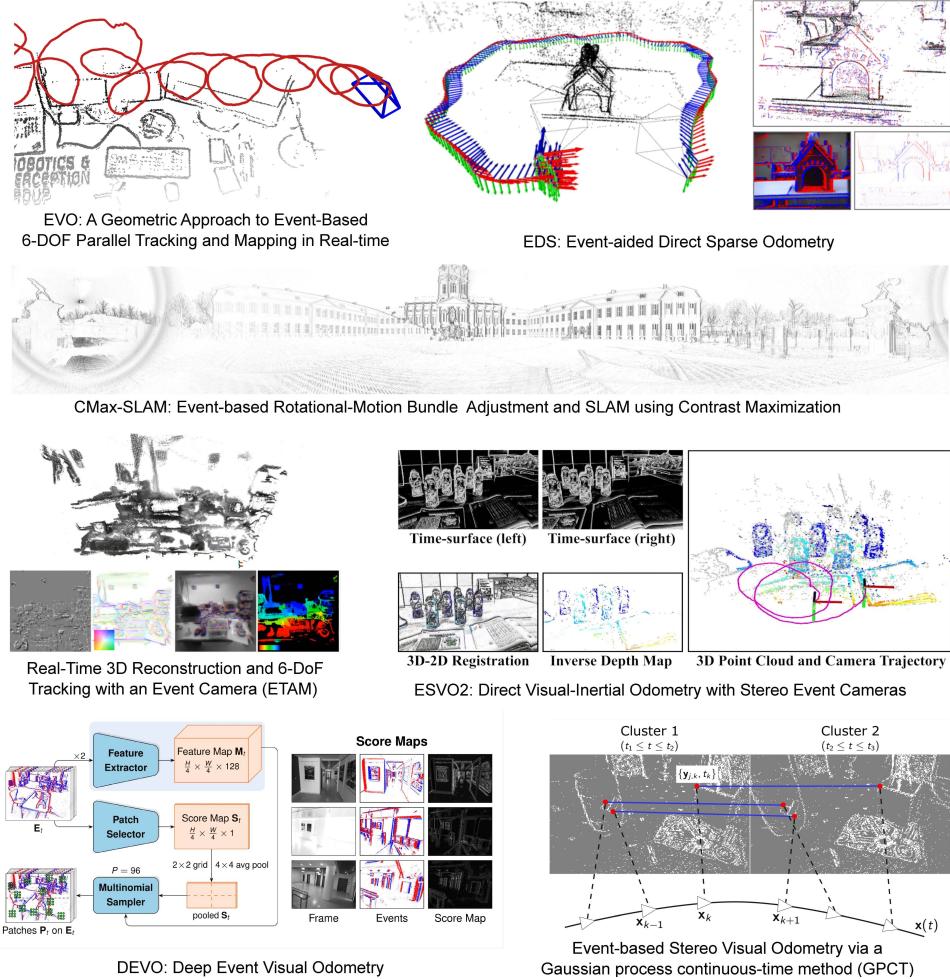


Figure 12.3 Event-based SLAM is actively being investigated, with systems that explore a large variety of approaches, including classical methods and more recent deep learning solutions. Since events are triggered by moving edges on the image plane, it is natural to recover scene maps in the form of edges (e.g., sparse or semi-dense 3D maps). Images adapted from EVO [592], EDS [311], CMax-SLAM [279], Kim et al. [382], ESVO2 [535], DEVO [391] and Wang et al. [737].

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.

## 12.4 Front-end

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. 12.4). The back-end refines these variables (i.e., the SLAM problem unknowns) to improve the fit between them and the sensor data. It 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. 12.3). 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.

### 12.4.1 Pre-processing. 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 [292] has been built). Many SLAM methods convert event data into alternative representations (event images, time maps or “time surfaces”, voxel grids, etc.) [245] for different reasons, such as compatibility with conventional computer

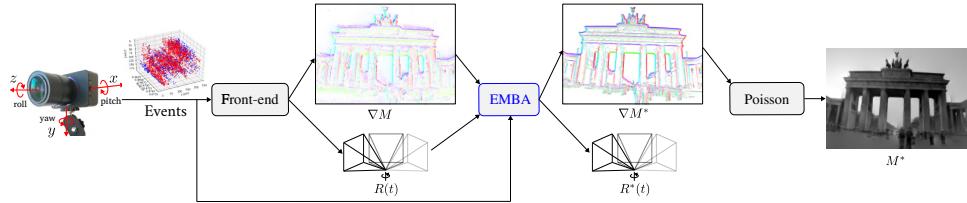


Figure 12.4 Event-based SLAM pipeline with a front-end (that computes and a map and camera poses) and a back-end (that refines the map and poses). Since events respond to moving edges, the recovered map is often an edge (gradient) map. The example shows a direct, rotational SLAM pipeline (poses consists of rotations, and the map reduces to a panoramic map) [280]. An absolute intensity map may be recovered by Poisson integration.

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 [245, 250] 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 [592] uses raw events for its mapping module (EMVS [594]) 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.

#### 12.4.2 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 focusing 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 [292]) 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).

#### 12.4.3 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, subsampling, etc.) are adopted to reduce the number of events to process [279, 383].

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, spatial event rate error, etc.). 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 corresponding 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.

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, dispersion, etc. They

are equivalently known as Focus or Contrast Maximization (CMax) [244, 653]. In problems where events can be warped to a few pixels or a line, these methods can suffer from undesired global optima [652]. 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.

#### **12.4.4 Model-based and Learning-based Methods**

So far, the majority of event-based SLAM approaches are hand-crafted, designed by human intuition on 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 [799]; whereas supervised methods require ground truth data for training [391], which is typically difficult to acquire in the real world. In recent years, many multi-sensor datasets have been recorded onboard cars, drones, etc., which can provide the data needed for ANN training.

Learning-based solutions may substitute parts of the SLAM pipeline, such as feature extraction and tracking [497], 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.

## **12.5 Back-end**

The goal of a refinement module like the SLAM back-end [90] 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. Often, bundle adjustment (BA) [709] is used as synonym for back-end.

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 [382, 592, 839], with each module relying on the output of the other concurrently running module as input to work properly. They 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 Sec. 12.2 (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 complexity and latency) and prone to local minima.

Only recently the problem of BA has been tackled 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 [709, 426, 519]. They operate on geometric primitives (corners, lines, etc.) that are detected in the event stream (possibly preceded by an events-to-image conversion [132, 609] 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 [292]. 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 [596, 824]) and it is not yet effective: 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 [406]. Examples of indirect back-ends include [609, 132, 737].

**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 [311], which leverage grayscale information from colocated frames, borrow the back-end from frame-based systems [207, 24]. 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 alignment (also called motion compensation or CMax) [243, 279] or the minimization of the photometric error (i.e., temporal contrast) conveyed by each event [280, 281]. They are designed based on the event generation model (12.1). 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.

## 12.6 State-of-the-Art Systems

Table 12.1 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.

System	M/DL	I/D	Event represent.	BA	Motion	Scene	Input	Remarks
Cook [149]	M	D	Event Frame	✗	Rot	Natural	E	Interacting network using optical flow
Weikersdorfer [750]	M	I	Individual Event	✗	Planar	2D B&W	E	First filter-based Ev-SLAM.
PF-SMT [381]	M	D	Individual Event	✗	Rot	Natural	E	Two interleaved Bayesian filters
Censi [108]	M	D	Event Packet	✗	6DoF	B&W	E+F+D	Filter-based VO based on image gradient
EB-SLAM-3D [751]	M	D	Individual Event	✗	6DoF	Natural	E+D	Augment events with depth sensor
Yuan [814]	M	I	Event Frame	✗	6DoF	B&W	E+I+M	Vertical line-based camera tracking
Kueng [406]	M	I	Local Point Set	✗	6DoF	Natural	E+F	Event-based feature tracking VO
ETAM [382]	M	D	Individual Events	✗	6DoF	Natural	E	Three interleaved filters
CMax - $\omega$ [240]	M	D	Individual Events	✗	Rot	Natural	E	Contrast Maximization
EVO [592]	M	D	Edge Map	✗	6DoF	Natural	E	Event-event geometric alignment
EVIO [840]	M	I	Point sets	✗	6DoF	Natural	E+I	Filter-based and MC features
Rebecq [593]	M	I	MC Event Images	✓	6DoF	Natural	E+I	Feature-based, sliding-window back-end
RTPT [599]	M	D	Individual Events	✗	Rot	Natural	E	Panoramic tracker and mapper
Gallego [242]	M	D	Individual Events	✗	6DoF	Natural	E+M	Resilient sensor model
Mueggler [510]	M	D	Individual Events	✓	6DoF	Natural	E+I+M	Continuous-time pose estimator
USLAM [609]	M	I	MC Event Images	✓	6DoF	Natural	E+F+I	Sensor fusion & sliding-window back-end
Chin [132]	M	I	Event Frames	✓	Rot	Stars	E	Tailored to star tracking
ESVO [839]	M	D	Time surfaces (TS)	✗	6DoF	Natural	2E	Stereo matching on TS patches
Hadviger [284]	M	I	Corners on TS	✗	6DoF	Natural	2E	Cross-corr. feature descriptors
CMax-GAE [383]	M	D	Individual Events	✗	Rot	Natural	E	Contrast maximization
EKLIT-VIO [479]	M	I	Individual events	✓	6DoF	Natural	E+F+I	EKLIT tracker and VIO back-end
EDS [311]	M	D	Event images	✓	6DoF	Natural	E+F	Frame-based back-end (DSO)
CB-VIO [452]	M	I	Individual events	✓	6DoF	Natural	E+F+I	Feature tracker and VIO back-end
Wang [737]	M	I	Binary images	✓	6DoF	Natural	2E	Feature matching
El Moudni [201]	M	D	Time Surfaces TS	✗	6DoF	Natural	2E	Use ESVO tracker and EMVS mapper
ESVIO [122]	M	I	Time surfaces (TS)	✓	6DoF	Natural	2E+2F+I	Feature tracking on from TS
ESVIO-direct [453]	M	D	Time surfaces (TS)	✓	6DoF	Natural	2E+I	Extension of ESVO
PL-EVIO [275]	M	I	Time surfaces (TS)	✓	6DoF	Natural	E+F+I	Point & line features, sliding-window BA
CMax-SLAM [279]	M	D	Individual Events	✓	Rot	Natural	E	Contrast Maximization refines motion
EVI-SAM [274]	M	D,I	Individual Events	✓	6DoF	Natural	E+F+I	Dense mapping
Zuo [849]	M	D	Individual Event	✗	6DoF	Natural	E+D	Augment events with depth sensor
DEVO [391]	DL	I	Event voxel grids	✓	6DoF	Natural	E	Event-version of DPVO [694]
EMBA [280]	M	D	Individual Events	✓	Rot	Natural	E	Refines motion and gradient map
EPBA [281]	M	D	Individual Events	✓	Rot	Natural	E	Refines motion and intensity map
ES-PTAM [261]	M	D	Events (also as frames)	✗	6DoF	Natural	2E	Use EVO tracker and EMVS mapper
ESVO2 [535]	M	D	Time surfaces (TS)	✓	6DoF	Natural	2E+I	Extension of ESVO
DEIO [273]	DL	I	Event voxel grids	✓	6DoF	Natural	E+I	Extension of DEVO and DPVO

Table 12.1 *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).*

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 im-

proved in an almost standard “exploitation” roadmap (similar to frame-based SLAM): for example, monocular methods [592] can be extended into stereo or multi-camera scenarios [261], event-only methods like [839] (resp. [391]) can be robustified using inertial data fusion [453, 535] (resp. [273]), 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 Tab. 12.1: 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.

## 12.7 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. Let us describe prominent SLAM datasets, benchmarks and simulators for event cameras.

### 12.7.1 Simulators

There are a variety of publicly available tools for generating high-quality synthetic event camera data. ESIM [595] is an evolved version of [509], which was one of the first simulators to mimic the principle of operation of an event camera. Previous efforts, such as [356], 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 [180] expands ESIM in more diverse, rich, and complex scenarios for autonomous driving. In the context of learning monocular depth from events [310], 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 12.5 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) [251]. 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.

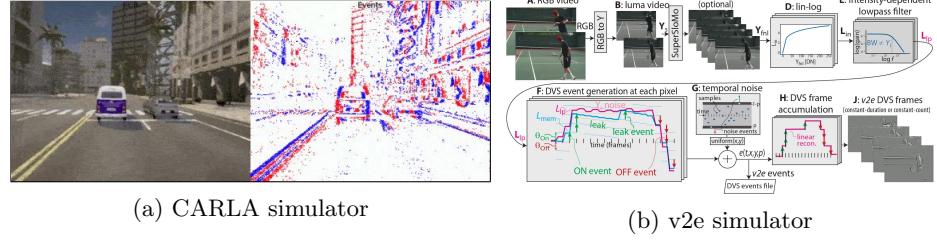


Figure 12.5 Event camera simulators: (a) RGB image and generated events using the ESIM simulator in CARLA [180]; (b) Detailed data processing steps of the V2E tool [826].

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 [347] 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.

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 [440]: the contrast threshold of an event camera ( $C$  in (12.1)) 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 [251] models an ideal event camera lighting. V2E [323] 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 [347] to increase the temporal resolution of the input video. Figure 12.5 depicts the V2E architecture in detail.

Simulating event camera noises is a challenging topic for realistic synthetic event generation, EventGan [843] 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 [604] 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 [26] 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) [826] 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.

### **12.7.2 Datasets and Benchmarks**

The number of event-based datasets dedicated to Visual Odometry and SLAM has increased significantly since the publication of the ECDS [509] (see Tab. 12.2). ECDS was the first dataset with synchronized events, IMU, and ground-truth camera poses in 6-DoF. Previous datasets [618] 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 [43] 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 [595] simulator.

The RPG stereo dataset [838] consists of eight hand-held sequences recorded with a stereo DAVIS [76] in an office environment and a synthetic sequence (featuring three fronto-parallel planes at various depths) produced by the simulator [509]. 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) [841] 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.

The UZH-FPV (First Person View) dataset [166] is specifically designed to advance research in autonomous drone racing. It features a custom-built quadrotor with a Qualcomm Flight Board and an mDAVIS346 [688] 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) [502] is the first event-based dataset created specifically for segmentation of independently moving objects (IMO) in indoor environments. It contains 32 minutes of recordings, tracking 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 [81] 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) [252] 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)

Dataset	Platforms	Pixel Resolution	Sensors
ECDS [509]	Hand-held	240 × 180	E, F, I
RPG-stereo [838]	Hand-held	240 × 180	2E
MVSEC [841]	Hand-held, Drone, Car, Byke	346 × 240	2E, 2F, I, Lidar, GPS
UZH-FPV [166]	Drone	346 × 260	E, F, I
EV-IMO [502]	Hand-held	346 × 260	E, F, I, Depth
EV-IMO2 [81]	Hand-held	640 × 480	3E, F, I, Depth
DSEC [252]	Car	640 × 480	2E, 2F, Lidar, GPS
TUM-VIE [390]	Hand-held	1280 × 720	2E, 2F, I
EDS [311]	Hand-held	640 × 480	E, F(RGB), I
VECtor [248]	Hand-held	640 × 480	2E, 2F, RGB-D, I, Lidar
M2DGR [807]	Ground Robot	640 × 480	E, F, I, Lidar, GPS, Thermal
ViViD++ [419]	Hand-held, Car	240 × 180, 640 × 480	E, F, RGB-D, Thermal, Lidar, GPS
FusionPortable [348]	Hand-held, Quadruped Robot	346 × 240	2E, 2F, I, Lidar, GPS
Stereo HKU-VIO [122]	Hand-held	346 × 260	2E, 2F, I
M3DE [111]	Drone, Car, Quadruped Robot	1280 × 720	2E, 2F, I, Lidar, GPS
CoSEC [564]	Car	1280 × 720	2E, 2F, I, Lidar, GPS

Table 12.2 *Summary of event-based SLAM datasets, sorted chronologically. Same notation for sensor data as in Tab. 12.1. Stereo and multi-sensor datasets are further described in the survey [260].*

with a 60 cm baseline and two RGB cameras (FLIR Blackfly S) with a 51 cm baseline (see Fig. 12.6). 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) [390] employs stereo Prophesee Gen4 event cameras (1 Megapixel resolution) along with synchronized 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 [311] 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. 12.6), 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

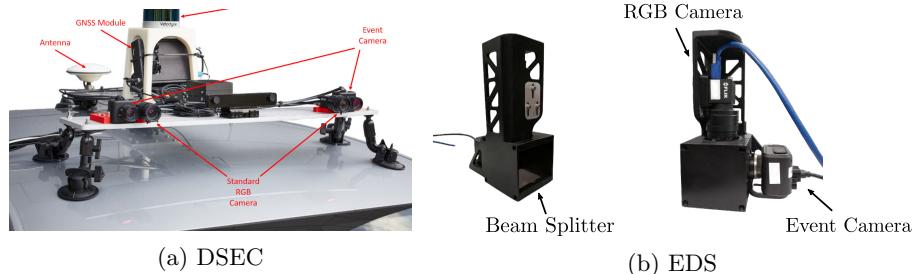


Figure 12.6 Details of some Event-SLAM datasets: (a) the sensor suite mounted on top of a car, in the DSEC [252] dataset. (b) Details of the beamsplitter that allows the sensors to share a spatially aligned field of view in the EDS [311] dataset.

like optical flow estimation, depth estimation, and robust visual odometry under various motion and lighting conditions.

The Versatile Event-Centric (VECtor) Benchmark Dataset [248] 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++) [419] 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 [348] 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) [111] (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.

### 12.7.3 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 *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) [673]. 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 [842]. 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 cameras, where abrupt and fast camera motions are estimated thanks to the events. Camera poses are functions of both velocities over time. Several toolboxes [272, 827] 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. 12.7). 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 [200, 310, 261].

## 12.8 Outlook

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 the performance (in terms of robustness, latency, efficiency, accuracy, etc.) 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 [555], 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 [157] 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 [797], spatially and temporally aligned, with low latency, HDR, and fine details (pixel count). Alternatively, foveated sensors [214], 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.

## Acknowledgment

The authors thank Giovanni Cioffi for his support in preparing this chapter.

# 13

## 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 13.1). Then, we introduce the concept of IMU preintegration (Section 13.2), 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 13.3). Finally, we showcase examples of what's achievable with modern IMU-centric SLAM systems (Section 13.4) and review recent trends (Section 13.5).

### 13.1 Basics of Inertial Sensing and Navigation

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 [113, 702]. 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.

### 13.1.1 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  $B$  of the robot, and the world frame  $W$  is an inertial frame.<sup>3</sup> The IMU measurements collected at time  $t$ , namely  $\dot{\mathbf{v}}(t)$  and  $\boldsymbol{\omega}(t)$ , are typically assumed to be corrupted by additive white Gaussian noise  $\boldsymbol{\eta}$  and slowly varying

<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) [215]. In robotics, this distinction is often de-emphasized in near-Earth small-scale applications where noisy low-cost IMUs are often used and the impact of the Earth rotation is negligible compared to the measurement noise. For this reason, the world frame  $W$ , which in robotics is typically chosen to be fixed at a location on Earth, is approximately treated as an inertial frame.

sensor biases  $\mathbf{b}$ :

$$\dot{\mathbf{v}}(t) = \mathbf{R}_w^B(t)^T (\dot{\mathbf{v}}^W(t) - \mathbf{g}^W) + \mathbf{b}^a(t) + \boldsymbol{\eta}^a(t), \quad (13.1)$$

$$\boldsymbol{\omega}(t) = \boldsymbol{\omega}_{WB}^B(t) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t). \quad (13.2)$$

As usual, the superscript B denotes that the corresponding quantity is expressed in the body (IMU) frame 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 B to W;  $\dot{\mathbf{v}}^W(t) \in \mathbb{R}^3$  is the acceleration of the sensor in the world frame;  $\mathbf{g}^W$  is the gravity vector in the world frame. Therefore, the term  $\mathbf{R}_w^B(t)(\dot{\mathbf{v}}^W(t) - \mathbf{g}^W)$  is the acceleration experienced by the IMU in the Body/IMU frame. The vector  $\boldsymbol{\omega}_{WB}^B(t) \in \mathbb{R}^3$  is the instantaneous angular velocity of B relative to W expressed in coordinate frame 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).

**Extended Models.** While the IMU measurement model (13.1)-(13.2) often suffices in robotics, more sophisticated models that more accurately model 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 (13.2) can be extended to:

$$\dot{\mathbf{v}}(t) = \mathbf{T}_a \mathbf{R}_w^B(t)^T (\dot{\mathbf{v}}^W(t) - \mathbf{g}^W) + \mathbf{b}^a(t) + \boldsymbol{\eta}^a(t), \quad (13.3)$$

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:

$$\boldsymbol{\omega}(t) = \mathbf{T}_g \boldsymbol{\omega}_{WB}^B(t) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t), \quad (13.4)$$

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:

$$\boldsymbol{\omega}(t) = \mathbf{T}_g \boldsymbol{\omega}_{WB}^B(t) + \mathbf{T}_s \mathbf{R}_w^B(t)^T (\dot{\mathbf{v}}^W(t) - \mathbf{g}^W) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t), \quad (13.5)$$

where  $\mathbf{T}_s$  is the g-sensitivity matrix, which can be estimated during calibration.

### 13.1.2 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.* (13.1)), we may 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 applications. However, we can determine the rotation corresponding to roll and pitch via the following static initialization:

$$\begin{cases} \mathbf{z}_W^B = \frac{\mathbf{g}^B}{\|\mathbf{g}^B\|} \\ \mathbf{x}_W^B = \frac{\mathbf{e}_1 - \mathbf{z}_W^B \mathbf{e}_1^\top \mathbf{z}_W^B}{\|\mathbf{e}_1 - \mathbf{z}_W^B \mathbf{e}_1^\top \mathbf{z}_W^B\|} \\ \mathbf{y}_W^B = \mathbf{z}_W^B \times \mathbf{x}_W^B \end{cases} \Rightarrow \mathbf{R}_W^B = [\mathbf{x}_W^B \quad \mathbf{y}_W^B \quad \mathbf{z}_W^B] \quad (13.6)$$

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. (13.6) 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 [215]), 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:

$$\begin{cases} \mathbf{g}^B = \mathbf{R}_W^B \mathbf{g}^W \\ \boldsymbol{\omega}_{ie}^B = \mathbf{R}_W^B \boldsymbol{\omega}_{ie}^W \\ \mathbf{g}^B \times \boldsymbol{\omega}_{ie}^B = \mathbf{R}_W^B (\mathbf{g}^W \times \boldsymbol{\omega}_{ie}^W) \end{cases} \Rightarrow \mathbf{R}_B^W = \left[ \begin{array}{c} \mathbf{g}^{W\top} \\ \boldsymbol{\omega}_{ie}^{W\top} \\ (\mathbf{g}^W \times \boldsymbol{\omega}_{ie}^W)^\top \end{array} \right]^{-1} \left[ \begin{array}{c} \mathbf{g}^{B\top} \\ \boldsymbol{\omega}_{ie}^{B\top} \\ (\mathbf{g}^B \times \boldsymbol{\omega}_{ie}^B)^\top \end{array} \right] \quad (13.7)$$

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 in practice.

### 13.2 IMU Preintegration and Factor Graphs

In the previous section, we have introduced the IMU measurement model (13.1)–(13.2), which relates the IMU measurements to the state of the robot, and in partic-

ular 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 2, 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 a quickly growing factor graph.

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. Intuitively, we can integrate IMU measurements over time to obtain relative motion measurements, and we can add these motion measurements to the factor graph instead. A naive integration of the IMU measurements (reviewed in Section 13.2.1) would still require repeating the integration of the measurements at each iteration of the factor graph solver. 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 [467] and has been extended to operate on manifold in [226, 225]; in Section 13.2.2, we closely follow the presentation in [226, 225]; then discuss more advanced preintegration techniques in Section 13.2.3. As usual, we postpone the discussion of recent works on the topic to Section 13.5.

### 13.2.1 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 [521, 215]:

$$\dot{\mathbf{R}}_B^W = \mathbf{R}_B^W (\boldsymbol{\omega}_{WB}^B)^\wedge, \quad \dot{\mathbf{v}}^W = \dot{\mathbf{v}}^W, \quad \dot{\mathbf{p}}^W = \mathbf{v}^W, \quad (13.8)$$

which describes the evolution of the rotation  $\mathbf{R}_B^W$ , translation  $\mathbf{p}^W$ , and velocity  $\mathbf{v}^W$  of the body frame B with respect to the world frame W.

The state at time  $t + \Delta t$ , where  $\Delta t$  is the IMU sampling period, is obtained by integrating (13.8):

$$\mathbf{R}_B^W(t + \Delta t) = \mathbf{R}_B^W(t) \text{Exp} \left( \int_t^{t+\Delta t} \boldsymbol{\omega}_{WB}^B(\tau) d\tau \right) \quad (13.9)$$

$$\begin{aligned} \mathbf{v}^W(t + \Delta t) &= \mathbf{v}^W(t) + \int_t^{t+\Delta t} \dot{\mathbf{v}}^W(\tau) d\tau \\ \mathbf{p}^W(t + \Delta t) &= \mathbf{p}^W(t) + \int_t^{t+\Delta t} \mathbf{v}^W(\tau) d\tau \end{aligned} \quad (13.10)$$

where in the first equation we assumed that the direction of the angular velocity

$\omega_{WB}^B$  does not change in the interval  $[t, t + \Delta t]$ .<sup>4</sup> Further assuming that  $\dot{\mathbf{v}}^W$  and  $\omega_{WB}^B$  remain constant in the time interval  $[t, t + \Delta t]$ , we can write:

$$\begin{aligned}\mathbf{R}_B^W(t + \Delta t) &= \mathbf{R}_B^W(t) \operatorname{Exp}(\omega_{WB}^B(t)\Delta t) \\ \mathbf{v}^W(t + \Delta t) &= \mathbf{v}^W(t) + \dot{\mathbf{v}}^W(t)\Delta t \\ \mathbf{p}^W(t + \Delta t) &= \mathbf{p}^W(t) + \mathbf{v}^W(t)\Delta t + \frac{1}{2}\dot{\mathbf{v}}^W(t)\Delta t^2.\end{aligned}\quad (13.11)$$

More generally, eq. (13.11) can be understood as applying Euler integration to numerically solve the integrals in (13.9). Using Eqs. (13.1)-(13.2), we can write  $\dot{\mathbf{v}}^W$  and  $\omega_{WB}^B$  as functions of the IMU measurements, hence (13.11) becomes

$$\begin{aligned}\mathbf{R}(t + \Delta t) &= \mathbf{R}(t) \operatorname{Exp}((\tilde{\boldsymbol{\omega}}(t) - \mathbf{b}^g(t) - \boldsymbol{\eta}^{gd}(t))\Delta t) \\ \mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \mathbf{g}\Delta t + \mathbf{R}(t)(\tilde{\mathbf{v}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t))\Delta t \\ \mathbf{p}(t + \Delta t) &= \mathbf{p}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 + \frac{1}{2}\mathbf{R}(t)(\tilde{\mathbf{v}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t))\Delta t^2,\end{aligned}\quad (13.12)$$

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 (13.8) 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 (13.12) as it is simple and amenable for modeling and uncertainty propagation, and then discuss more advanced integration techniques in Section 13.2.3. 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  $\operatorname{Cov}(\boldsymbol{\eta}^{gd}(t)) = \frac{1}{\Delta t}\operatorname{Cov}(\boldsymbol{\eta}^g(t))$ . The same relation holds for  $\boldsymbol{\eta}^{ad}$  (cf., [150, Appendix]).

While Eq. (13.12) 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. (13.12) 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 [335].

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 9), we can use the expression (13.12) 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 (13.12) for all  $\Delta t$  intervals

<sup>4</sup> We provide a more general expression for the rotation integration in eq. (13.35) below.

<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*. However, this term

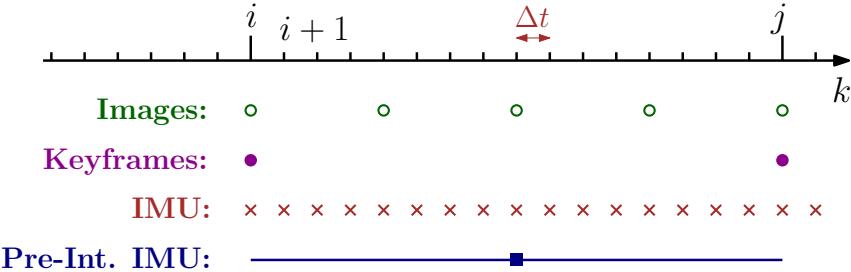


Figure 13.1 Different rates for IMU and camera.

between two consecutive keyframes at times  $t_i$  and  $t_j$  (*c.f.*, Fig. 13.1), we get:<sup>6</sup>

$$\begin{aligned} \mathbf{R}_j &= \mathbf{R}_i \prod_{k=i}^{j-1} \text{Exp} \left( \left( \tilde{\boldsymbol{\omega}}_k - \mathbf{b}_k^g - \boldsymbol{\eta}_k^{gd} \right) \Delta t \right), \\ \mathbf{v}_j &= \mathbf{v}_i + \mathbf{g} \Delta t_{ij} + \sum_{k=i}^{j-1} \mathbf{R}_k \left( \tilde{\mathbf{v}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t \\ \mathbf{p}_j &= \mathbf{p}_i + \sum_{k=i}^{j-1} \left[ \mathbf{v}_k \Delta t + \frac{1}{2} \mathbf{g} \Delta t^2 + \frac{1}{2} \mathbf{R}_k \left( \tilde{\mathbf{v}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t^2 \right] \end{aligned} \quad (13.13)$$

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. (13.13) already provides an estimate of the motion between time  $t_i$  and  $t_j$ , it has the drawback that the integration in (13.13) has to be repeated whenever the linearization point at time  $t_i$  changes [426] (*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 (13.13).

### 13.2.2 IMU Preintegration on Manifold

Here we show that a small manipulation of the motion integration results (13.13) 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

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 13.4.3 for further discussion about temporal synchronization.

leads to computing the so called *preintegrated IMU measurements*, which constrain the motion between consecutive states in the factor graph.

Towards this goal, we define the following relative motion increments that are independent of the pose and velocity at  $t_i$ :

$$\begin{aligned}\Delta \mathbf{R}_{ij} &\doteq \mathbf{R}_i^\top \mathbf{R}_j = \prod_{k=i}^{j-1} \text{Exp} \left( \left( \tilde{\boldsymbol{\omega}}_k - \mathbf{b}_k^g - \boldsymbol{\eta}_k^{gd} \right) \Delta t \right) \\ \Delta \mathbf{v}_{ij} &\doteq \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g} \Delta t_{ij}) = \sum_{k=i}^{j-1} \Delta \mathbf{R}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t \\ \Delta \mathbf{p}_{ij} &\doteq \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2) \\ &= \sum_{k=i}^{j-1} \left[ \Delta \mathbf{v}_{ik} \Delta t + \frac{1}{2} \Delta \mathbf{R}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad} \right) \Delta t^2 \right],\end{aligned}\quad (13.14)$$

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 (13.14) independent from the state at time  $i$  as well as gravitational effects. Indeed, we will be able to compute the right-hand side of (13.14) directly from the inertial measurements between the two keyframes.

Unfortunately, summations and products in (13.14) are still function of the bias estimate. We tackle this problem in two steps. In Section 13.2.2.1, we assume  $\mathbf{b}_i$  is known; then, in Section 13.2.2.3 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  $t_j$ :

$$\mathbf{b}_i^g = \mathbf{b}_{i+1}^g = \dots = \mathbf{b}_{j-1}^g, \quad \mathbf{b}_i^a = \mathbf{b}_{i+1}^a = \dots = \mathbf{b}_{j-1}^a. \quad (13.15)$$

### 13.2.2.1 Preintegrated IMU Measurements

Equation (13.14) 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 (13.14) so to make easier the derivation of the measurement log-likelihood. More concretely, we isolate the noise terms of the individual inertial measurements in (13.14). 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 (13.14). Towards this goal, we

use the following properties of the exponential map for SO(3):

$$\text{Exp}(\phi + \delta\phi) \approx \text{Exp}(\phi) \text{Exp}(J_r(\phi)\delta\phi), \quad (13.16)$$

$$\text{Exp}(\phi) \mathbf{R} = \mathbf{R} \text{Exp}(\mathbf{R}^\top \phi). \quad (13.17)$$

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 (13.16) and (13.17), we rearrange the terms in the expression of  $\Delta\mathbf{R}_{ij}$  in (13.14), by “moving” the noise to the end:

$$\begin{aligned} \Delta\mathbf{R}_{ij} &\stackrel{\text{eq.(13.16)}}{\simeq} \prod_{k=i}^{j-1} \left[ \text{Exp}((\tilde{\omega}_k - \mathbf{b}_i^g) \Delta t) \text{Exp}\left(-J_r^k \boldsymbol{\eta}_k^{gd} \Delta t\right) \right] \\ &\stackrel{\text{eq.(13.17)}}{=} \Delta\tilde{\mathbf{R}}_{ij} \prod_{k=i}^{j-1} \text{Exp}\left(-\Delta\tilde{\mathbf{R}}_{k+1j}^\top J_r^k \boldsymbol{\eta}_k^{gd} \Delta t\right) \\ &\doteq \Delta\tilde{\mathbf{R}}_{ij} \text{Exp}(-\delta\phi_{ij}) \end{aligned} \quad (13.18)$$

with  $J_r^k \doteq J_r^k((\tilde{\omega}_k - \mathbf{b}_i^g)\Delta t)$ . In the last line of (13.18), we defined the *preintegrated rotation measurement*  $\Delta\tilde{\mathbf{R}}_{ij} \doteq \prod_{k=i}^{j-1} \text{Exp}((\tilde{\omega}_k - \mathbf{b}_i^g) \Delta t)$ , 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 (13.14) by using the following relations:

$$\exp(\phi^\wedge) \approx \mathbf{I} + \phi^\wedge, \quad (13.19)$$

$$\mathbf{a}^\wedge \mathbf{b} = -\mathbf{b}^\wedge \mathbf{a}, \quad \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^3, \quad (13.20)$$

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 (13.18) back into the expression of  $\Delta\mathbf{v}_{ij}$  in (13.14), using the first-order approximation (13.19) for  $\text{Exp}(-\delta\phi_{ij})$ , and dropping higher-order noise terms, we obtain:

$$\begin{aligned} \Delta\mathbf{v}_{ij} &\stackrel{\text{eq.(13.19)}}{\simeq} \sum_{k=i}^{j-1} \Delta\tilde{\mathbf{R}}_{ik} (\mathbf{I} - \delta\phi_{ik}^\wedge) \left( \tilde{\mathbf{v}}_k - \mathbf{b}_i^a \right) \Delta t - \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t \\ &\stackrel{\text{eq.(13.20)}}{=} \Delta\tilde{\mathbf{v}}_{ij} + \sum_{k=i}^{j-1} \left[ \Delta\tilde{\mathbf{R}}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_i^a \right)^\wedge \delta\phi_{ik} \Delta t - \Delta\tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t \right] \\ &\doteq \Delta\tilde{\mathbf{v}}_{ij} - \delta\mathbf{v}_{ij} \end{aligned} \quad (13.21)$$

where we defined the *preintegrated velocity measurement*  $\Delta\tilde{\mathbf{v}}_{ij} \doteq \sum_{k=i}^{j-1} \Delta\tilde{\mathbf{R}}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_i^a \right) \Delta t$  and its noise  $\delta\mathbf{v}_{ij}$ .

Similarly, substituting (13.18) and (13.21) in the expression of  $\Delta\mathbf{p}_{ij}$  in (13.14),

and using the first-order approximation (13.19), we obtain:

$$\begin{aligned} \Delta \mathbf{p}_{ij} &\stackrel{\text{eq. (13.19)}}{\simeq} \sum_{k=i}^{j-1} \left[ (\Delta \tilde{\mathbf{v}}_{ik} - \delta \mathbf{v}_{ik}) \Delta t + \frac{1}{2} \Delta \tilde{\mathbf{R}}_{ik} (\mathbf{I} - \delta \phi_{ik}^{\wedge}) (\tilde{\mathbf{v}}_k - \mathbf{b}_i^a) \Delta t^2 \right. \\ &\quad \left. - \frac{1}{2} \Delta \tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t^2 \right] \\ &\stackrel{\text{eq. (13.20)}}{=} \Delta \tilde{\mathbf{p}}_{ij} + \sum_{k=i}^{j-1} \left[ -\delta \mathbf{v}_{ik} \Delta t + \frac{1}{2} \Delta \tilde{\mathbf{R}}_{ik} (\tilde{\mathbf{v}}_k - \mathbf{b}_i^a)^{\wedge} \delta \phi_{ik} \Delta t^2 \right. \\ &\quad \left. - \frac{1}{2} \Delta \tilde{\mathbf{R}}_{ik} \boldsymbol{\eta}_k^{ad} \Delta t^2 \right] \\ &\doteq \Delta \tilde{\mathbf{p}}_{ij} - \delta \mathbf{p}_{ij}, \end{aligned} \quad (13.22)$$

where we defined the *preintegrated position measurement*  $\Delta \tilde{\mathbf{p}}_{ij}$  and its noise  $\delta \mathbf{p}_{ij}$ .

Substituting the expressions (13.18), (13.21), (13.22) back in the original definition of  $\Delta \mathbf{R}_{ij}, \Delta \mathbf{v}_{ij}, \Delta \mathbf{p}_{ij}$  in (13.14), we finally get our *preintegrated measurement model* (remember  $\text{Exp}(-\delta \phi_{ij})^T = \text{Exp}(\delta \phi_{ij})$ ):

$$\begin{aligned} \Delta \tilde{\mathbf{R}}_{ij} &= \mathbf{R}_i^T \mathbf{R}_j \text{Exp}(\delta \phi_{ij}) \\ \Delta \tilde{\mathbf{v}}_{ij} &= \mathbf{R}_i^T (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g} \Delta t_{ij}) + \delta \mathbf{v}_{ij} \\ \Delta \tilde{\mathbf{p}}_{ij} &= \mathbf{R}_i^T \left( \mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2 \right) + \delta \mathbf{p}_{ij} \end{aligned} \quad (13.23)$$

where our compound measurements are written as a function of the (to-be-estimated) state “plus” a random noise, described by the random vector  $[\delta \phi_{ij}^T, \delta \mathbf{v}_{ij}^T, \delta \mathbf{p}_{ij}^T]^T$ .

In summary, in this section we manipulated the measurement model (13.14) and rewrote it as (13.23). The advantage of the measurements in eq. (13.23) 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.

### 13.2.2.2 Noise Propagation

In this section we derive the statistics of the noise vector  $[\delta \phi_{ij}^T, \delta \mathbf{v}_{ij}^T, \delta \mathbf{p}_{ij}^T]^T$ . 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  $\Sigma_{ij}$  of the preintegrated measurements:

$$\boldsymbol{\eta}_{ij}^{\Delta} \doteq [\delta \phi_{ij}^T, \delta \mathbf{v}_{ij}^T, \delta \mathbf{p}_{ij}^T]^T \sim \mathcal{N}(\mathbf{0}_{9 \times 1}, \Sigma_{ij}). \quad (13.24)$$

We first consider the preintegrated rotation noise  $\delta \phi_{ij}$ . Recall from (13.18) that

$$\text{Exp}(-\delta \phi_{ij}) \doteq \prod_{k=i}^{j-1} \text{Exp} \left( -\Delta \tilde{\mathbf{R}}_{k+1j}^T \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t \right). \quad (13.25)$$

Taking the Log on both sides and changing signs, we get:

$$\delta \phi_{ij} = -\text{Log} \left( \prod_{k=i}^{j-1} \text{Exp} \left( -\Delta \tilde{\mathbf{R}}_{k+1j}^T \mathbf{J}_r^k \boldsymbol{\eta}_k^{gd} \Delta t \right) \right). \quad (13.26)$$

Next, we use the following first-order approximation holds for SO(3) logarithm:

$$\text{Log}(\text{Exp}(\phi) \text{Exp}(\delta\phi)) \approx \phi + J_r^{-1}(\phi)\delta\phi. \quad (13.27)$$

where  $J_r^{-1}(\phi)$  is the inverse of the right Jacobian. Repeated application of (13.27) (recall that  $\eta_k^{gd}$  as well as  $\delta\phi_{ij}$  are small rotation noises, hence the right Jacobians are close to the identity) produces:

$$\delta\phi_{ij} \simeq \sum_{k=i}^{j-1} \Delta\tilde{R}_{k+1j}^T J_r^k \eta_k^{gd} \Delta t \quad (13.28)$$

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  $\eta_k^{gd}$ .

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  $\eta_k^{ad}$  and the preintegrated rotation noise  $\delta\phi_{ij}$ , hence they are also zero-mean and Gaussian. Simple manipulation leads to:

$$\begin{aligned} \delta\mathbf{v}_{ij} &\simeq \sum_{k=i}^{j-1} \left[ -\Delta\tilde{R}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_i^a \right)^{\wedge} \delta\phi_{ik} \Delta t + \Delta\tilde{R}_{ik} \eta_k^{ad} \Delta t \right] \\ \delta\mathbf{p}_{ij} &\simeq \sum_{k=i}^{j-1} \left[ \delta\mathbf{v}_{ik} \Delta t - \frac{1}{2} \Delta\tilde{R}_{ik} \left( \tilde{\mathbf{v}}_k - \mathbf{b}_i^a \right)^{\wedge} \delta\phi_{ik} \Delta t^2 + \frac{1}{2} \Delta\tilde{R}_{ik} \eta_k^{ad} \Delta t^2 \right] \end{aligned} \quad (13.29)$$

where the relations are valid up to the first order.

Eqs. (13.28)-(13.29) express the preintegrated noise  $\eta_{ij}^{\Delta}$  as a linear function of the IMU measurement noise  $\eta_k^d \doteq [\eta_k^{gd}, \eta_k^{ad}]$ ,  $k = 1, \dots, j-1$ . Therefore, from the knowledge of the covariance of  $\eta_k^d$  (given in the IMU specifications), we can compute the covariance of  $\eta_{ij}^{\Delta}$ , namely  $\Sigma_{ij}$ , by a simple linear propagation.

An extended derivation of the noise propagation can be found in [225], 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.

### 13.2.2.3 Incorporating Bias Updates

In the previous section, we assumed that the bias  $\{\bar{\mathbf{b}}_i^a, \bar{\mathbf{b}}_i^g\}$  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:

$$\begin{aligned} \Delta\tilde{R}_{ij}(\mathbf{b}_i^g) &\simeq \Delta\tilde{R}_{ij}(\bar{\mathbf{b}}_i^g) \text{Exp}\left(\frac{\partial\Delta\tilde{R}_{ij}}{\partial\mathbf{b}^g} \delta\mathbf{b}^g\right) \\ \Delta\tilde{\mathbf{v}}_{ij}(\mathbf{b}_i^g, \mathbf{b}_i^a) &\simeq \Delta\tilde{\mathbf{v}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g} \delta\mathbf{b}_i^g + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a} \delta\mathbf{b}_i^a \\ \Delta\tilde{\mathbf{p}}_{ij}(\mathbf{b}_i^g, \mathbf{b}_i^a) &\simeq \Delta\tilde{\mathbf{p}}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^g} \delta\mathbf{b}_i^g + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^a} \delta\mathbf{b}_i^a \end{aligned} \quad (13.30)$$

This is similar to the bias correction in [467] but operates directly on SO(3). The Jacobians  $\{\frac{\partial \Delta \tilde{R}_{ij}}{\partial \mathbf{b}^g}, \frac{\partial \Delta \tilde{v}_{ij}}{\partial \mathbf{b}^g}, \dots\}$  (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 13.2.2.1 to express the measurements as a large value *plus* a small perturbation and is given in [225].

#### 13.2.2.4 Preintegrated IMU Factors and Bias Models

Given the preintegrated measurement model in (13.23) and since measurement noise is zero-mean and Gaussian (with covariance  $\Sigma_{ij}$ ) up to first order (13.24), it is now easy to write the residual errors  $\mathbf{r}_{I_{ij}} \doteq [\mathbf{r}_{\Delta R_{ij}}^\top, \mathbf{r}_{\Delta v_{ij}}^\top, \mathbf{r}_{\Delta p_{ij}}^\top]^\top \in \mathbb{R}^9$ , which will appear in the factor graph optimization:

$$\begin{aligned} \mathbf{r}_{\Delta R_{ij}} &\doteq \text{Log} \left( \left( \Delta \tilde{R}_{ij}(\bar{\mathbf{b}}_i^g) \text{Exp} \left( \frac{\partial \Delta \tilde{R}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g \right) \right)^\top \mathbf{R}_i^\top \mathbf{R}_j \right) \\ \mathbf{r}_{\Delta v_{ij}} &\doteq \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g} \Delta t_{ij}) \\ &\quad - \left[ \Delta \tilde{v}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{v}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g + \frac{\partial \Delta \tilde{v}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}^a \right] \\ \mathbf{r}_{\Delta p_{ij}} &\doteq \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{ij} - \frac{1}{2} \mathbf{g} \Delta t_{ij}^2) \\ &\quad - \left[ \Delta \tilde{p}_{ij}(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a) + \frac{\partial \Delta \tilde{p}_{ij}}{\partial \mathbf{b}^g} \delta \mathbf{b}^g + \frac{\partial \Delta \tilde{p}_{ij}}{\partial \mathbf{b}^a} \delta \mathbf{b}^a \right], \end{aligned} \quad (13.31)$$

in which we also included the bias updates of Eq. (13.30). These terms can be readily added to the factor graph by adding the term  $\|\mathbf{r}_{I_{ij}}\|_{\Sigma_{ij}}^2$  to the objective of the minimization.

When presenting the IMU model (13.1)-(13.2), we said that biases are slowly time-varying quantities. Hence, we model them with a “Brownian motion”, *i.e.*, integrated white noise:

$$\dot{\mathbf{b}}^g(t) = \boldsymbol{\eta}^{bg}, \quad \dot{\mathbf{b}}^a(t) = \boldsymbol{\eta}^{ba}. \quad (13.32)$$

Integrating (13.32) over the time interval  $[t_i, t_j]$  between two consecutive keyframes  $i$  and  $j$  we get:

$$\mathbf{b}_j^g = \mathbf{b}_i^g + \boldsymbol{\eta}^{bgd}, \quad \mathbf{b}_j^a = \mathbf{b}_i^a + \boldsymbol{\eta}^{bad}, \quad (13.33)$$

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  $\Sigma^{bgd} \doteq \Delta t_{ij} \text{Cov}(\boldsymbol{\eta}^{bg})$  and  $\Sigma^{bad} \doteq \Delta t_{ij} \text{Cov}(\boldsymbol{\eta}^{ba})$ , respectively (*cf.* [150, Appendix]).

The model (13.33) can be readily included in our factor graph, as a further additive term in the objective function, for all consecutive keyframes:

$$\|\mathbf{r}_{\mathbf{b}_{ij}}\|^2 \doteq \|\mathbf{b}_j^g - \mathbf{b}_i^g\|_{\Sigma^{bgd}}^2 + \|\mathbf{b}_j^a - \mathbf{b}_i^a\|_{\Sigma^{bad}}^2 \quad (13.34)$$

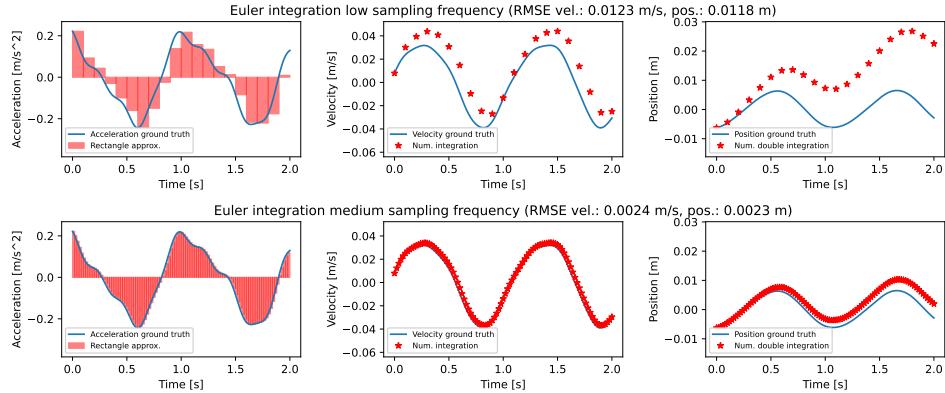


Figure 13.2 Example of Euler integration (with known initial conditions) using low (top row) and high (bottom row) sampling frequencies.

### 13.2.3 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 (13.14). 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.

#### 13.2.3.1 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 13.2 (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.

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 13.2 (top)). A possible workaround consists in increasing the sampling frequency of the signal (Figure 13.2 (bottom)). However, in real-world inertial navigation problems, the sampling frequency is limited by the hardware characteristics of the inertial sensor.

In [415], 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.

### 13.2.3.2 Continuous Acceleration Preintegration

Another way to reduce the integration error is to leverage continuous 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 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 representations while assuming solved rotation integration. Continuous integration over the rotation space will be addressed in the following subsection.

In [198], after using a zeroth-order integrator [707] 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 [198]). Compared with the standard preintegration [227] that consider constant global acceleration, the work [198] 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 [86] over both the standard preintegration and the constant accelerometer measurement model.

In order to loosen the constant acceleration motion model assumptions, one can approximate the input data with analytically integrable functions. Assuming that the rotational part of the preintegration is solved, the authors in [416] represent the rotation-corrected accelerometer measurements  $\hat{\mathbf{v}}_k$ , defined as  $\hat{\mathbf{v}}_k = \Delta\mathbf{R}_{ik}\hat{\mathbf{v}}_k$ , in a continuous manner. We show in Figure 13.3 the accuracy gain of integration with both piecewise-linear and GP-based continuous representations compared to the Euler method shown in Figure 13.2. With the piecewise-linear approximation, the first integral (from  $\hat{\mathbf{v}}_k$  to  $\Delta\mathbf{v}_{ik}$ ) corresponds to the classic trapezoidal rule

<sup>7</sup> GP regression is a non-parametric, probabilistic approach for interpolation. We invite the reader to refer to [590] for a deeper understanding of GP regression.

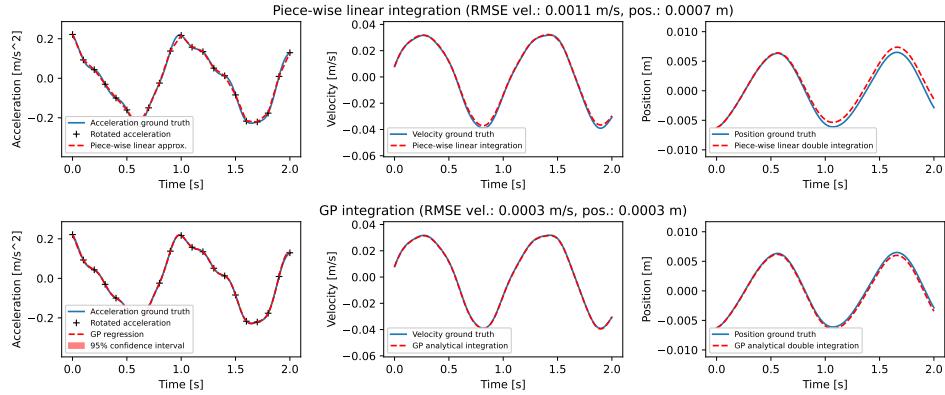


Figure 13.3 Top row: Continuous integration with piecewise-linear approximation (corresponding to constant-jerk motion assumption). Bottom row: Model-free integration with Gaussian Process regression.

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  $\hat{\mathbf{v}} \sim \mathcal{GP}(0, k_{\hat{\mathbf{v}}}(t, t')\mathbf{I})$  and  $k_{\hat{\mathbf{v}}}(t, t')$  is the square exponential covariance kernel function, the direct analytical inference of the integral (and double integral) of  $\hat{\mathbf{v}}$  is enabled by the application of linear operators on GPs [625]. Accordingly, the method does not rely on any explicit motion model as the square exponential kernel is infinitely differentiable. The bottom row of Figure 13.3 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.

#### 13.2.3.3 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

$$\mathbf{R}_{\text{B}}^{\text{W}}(t + \Delta t) = \mathbf{R}_{\text{B}}^{\text{W}}(t) \prod_t^{t + \Delta t} \text{Exp}(\boldsymbol{\omega}_{\text{WB}}^{\text{B}}(\tau))^{d\tau} \quad (13.35)$$

that solves the kinematic model (13.8) does not have a known generic solution [74] and novel approaches are required to perform continuous model-less integration over the space of rotations.

In response to this challenge, the authors of [414] 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

$$\dot{\mathbf{r}} = (\mathbf{J}_r(\mathbf{r}))^{-1} \boldsymbol{\omega}_{\text{WB}}^{\text{B}}, \quad (13.36)$$

where  $\mathbf{J}_r(\mathbf{r})$  is the right-hand Jacobian of  $\text{SO}(3)$  evaluated at  $\mathbf{r}$ . Unfortunately, neither  $\mathbf{r}$  nor  $\dot{\mathbf{r}}$  are directly observed by the IMU. The key idea of [414] is to model  $\dot{\mathbf{r}}$  with a GP and a set of virtual observations  $\dot{\mathbf{r}}_{t_\bullet}$  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 (13.36) and the gyroscope measurements as observations of  $\boldsymbol{\omega}_{\text{WB}}^{\text{B}}$ . 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 [35] 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 [257] 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.

### 13.3 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 [77, 303].

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 [327, 325]. 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 [793, 306, 434], determine the minimal measurements needed to initialize an estimator [306, 486], and also identify degenerate motions that cause additional unobservable directions and should be avoided or alerted if possible in practice [794]. For these reasons, significant research efforts have been devoted to the observability analysis of AINS [793], and in particular visual-inertial systems (*e.g.*, [307, 435, 794]).

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 13.3.1 introduces linearized models assuming exteroceptive measurements of geometric landmarks, Section 13.3.2 uses these models to perform the observability analysis, and Section 13.3.3 discusses degenerate configurations.

### 13.3.1 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.*, [306, 434, 429, 581, 256, 225]), line and plane features can be utilized when available (*e.g.*, [398, 305, 277, 795]). 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):

$$\mathbf{x} = \{\mathbf{R}_B^W, \mathbf{b}^g, \mathbf{v}^W, \mathbf{b}^a, \mathbf{p}^W, \mathbf{x}_f^W\} \quad (13.37)$$

where  $\mathbf{R}_B^W$  is the rotation of the body frame B with respect to the world frame 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.

### 13.3.1.1 Linearized IMU Kinematic Model

The IMU-based kinematic model is given by (*cf.* (13.8) and (13.32)):

$$\dot{\mathbf{R}}_B^W = \mathbf{R}_B^W(\boldsymbol{\omega}_{WB}^B)^\wedge, \quad \dot{\mathbf{v}}^W = \dot{\mathbf{v}}^W, \quad \dot{\mathbf{p}}^W = \mathbf{v}^W, \quad (13.38)$$

$$\dot{\mathbf{b}}^g(t) = \boldsymbol{\eta}^{bg}, \quad \dot{\mathbf{b}}^a(t) = \boldsymbol{\eta}^{ba} \quad (13.39)$$

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:

$$\dot{\tilde{\mathbf{x}}}(t) \simeq \begin{bmatrix} \mathbf{F}_c(t) & \mathbf{0}_{15 \times n_f} \\ \mathbf{0}_{n_f \times 15} & \mathbf{0}_{n_f} \end{bmatrix} \tilde{\mathbf{x}}(t) + \begin{bmatrix} \mathbf{G}_c(t) \\ \mathbf{0}_{n_f \times 12} \end{bmatrix} \boldsymbol{\eta}(t) =: \mathbf{F}(t)\tilde{\mathbf{x}}(t) + \mathbf{G}(t)\boldsymbol{\eta}(t) \quad (13.40)$$

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 (13.40),  $n_f$  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 (13.38) with the accelerometer and gyroscope measurements (*cf.* with derivation in Section 13.2.1).

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  $\dot{\Phi}_{(k+1,k)} = \mathbf{F}(t_k)\Phi_{(k+1,k)}$  with identity as the initial condition:

$$\Phi_{(k+1,k)} = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \Phi_{31} & \Phi_{32} & \mathbf{I}_3 & \Phi_{34} & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_{n_f \times 3} \\ \Phi_{51} & \Phi_{52} & \Phi_{53} & \Phi_{54} & \mathbf{I}_3 & \mathbf{0}_{n_f \times 3} \\ \mathbf{0}_{3 \times n_f} & \mathbf{I}_{n_f} \end{bmatrix} \quad (13.41)$$

where the  $(i,j)$  block  $\Phi_{ij}$  can be found analytically or numerically [306].

<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:  $\mathbf{R}_B^W = \hat{\mathbf{R}}_B^W \mathbf{R}_B^W(\tilde{\boldsymbol{\theta}}) \simeq \hat{\mathbf{R}}_B^W(\mathbf{I} + \tilde{\boldsymbol{\theta}}^\wedge)$ .

### 13.3.1.2 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.

**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 C:

$$\mathbf{z}_p = \underbrace{\begin{bmatrix} \lambda_r & \mathbf{0}_{1 \times 2} \\ \mathbf{0}_{2 \times 1} & \lambda_b \mathbf{I}_2 \end{bmatrix}}_{\boldsymbol{\Lambda}} \begin{bmatrix} z_r \\ z_b \end{bmatrix} = \boldsymbol{\Lambda} \begin{bmatrix} \|\mathbf{p}_f^C\| + \eta^r \\ h_b(\mathbf{p}_f^C) + \boldsymbol{\eta}^b \end{bmatrix} \quad (13.42)$$

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  $z_b$  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.  $\boldsymbol{\Lambda}$  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 (13.42) with the chain rule of differentiation at the current state estimate yields the following measurement error equation:

$$\tilde{\mathbf{z}}_p = \mathbf{z}_p - \hat{\mathbf{z}}_p \simeq \boldsymbol{\Lambda} \left[ \begin{array}{c|c} \frac{\partial z_r}{\partial \mathbf{p}_f^C} \frac{\partial \mathbf{p}_f^C}{\partial \hat{\mathbf{x}}} & \tilde{\mathbf{x}} + \eta^r \\ \frac{\partial z_b}{\partial \mathbf{p}_f^C} \frac{\partial \mathbf{p}_f^C}{\partial \hat{\mathbf{x}}} & \tilde{\mathbf{x}} + \boldsymbol{\eta}^b \end{array} \right] =: \boldsymbol{\Lambda} \begin{bmatrix} \mathbf{H}_r \\ \mathbf{H}_b \end{bmatrix} \mathbf{H}_f \tilde{\mathbf{x}} + \boldsymbol{\Lambda} \begin{bmatrix} \eta^r \\ \boldsymbol{\eta}^b \end{bmatrix} =: \mathbf{H}_x \tilde{\mathbf{x}} + \boldsymbol{\eta}^p \quad (13.43)$$

where  $\hat{\mathbf{z}}_p$  is the measurement at the linearization point. Depending on the selection matrix  $\boldsymbol{\Lambda}$ , 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 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:

$$\mathbf{l}^W = \begin{bmatrix} \mathbf{n}_\ell^W \\ \mathbf{v}_\ell^W \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^W \times \mathbf{p}_2^W \\ \mathbf{p}_2^W - \mathbf{p}_1^W \end{bmatrix} \quad (13.44)$$

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 [657]:

$$\begin{bmatrix} \mathbf{n}_\ell^C \\ \mathbf{v}_\ell^C \end{bmatrix} = \begin{bmatrix} \mathbf{R}_c^{W^\top} & -\mathbf{R}_c^{W^\top}(\mathbf{p}_c^W)^\wedge \\ \mathbf{0} & \mathbf{R}_c^{W^\top} \end{bmatrix} \begin{bmatrix} \mathbf{n}_\ell^W \\ \mathbf{v}_\ell^W \end{bmatrix} \quad (13.45)$$

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 [848]. To this end, we transform the 3D line from the world frame to the current camera frame via (13.45) and then project it onto the image with the known intrinsic parameters of the camera [657]:

$$\boldsymbol{\ell} = \underbrace{\begin{bmatrix} f_2 & 0 & 0 \\ 0 & f_1 & 0 \\ -f_2 c_1 & -f_1 c_2 & f_1 f_2 \end{bmatrix}}_{\mathbf{K}} [\mathbf{I}_3 \quad \mathbf{0}_3] \begin{bmatrix} \mathbf{n}_{\ell}^C \\ \mathbf{v}_{\ell}^C \end{bmatrix} =: \begin{bmatrix} \ell_1 \\ \ell_2 \\ \ell_3 \end{bmatrix} \quad (13.46)$$

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:

$$\mathbf{z}_{\ell} = \begin{bmatrix} \frac{\mathbf{q}_1^\top \boldsymbol{\ell}}{\sqrt{\ell_1^2 + \ell_2^2}} \\ \frac{\mathbf{q}_2^\top \boldsymbol{\ell}}{\sqrt{\ell_1^2 + \ell_2^2}} \end{bmatrix} + \boldsymbol{\eta}^{\ell} \quad (13.47)$$

where  $\boldsymbol{\eta}^{\ell}$  is the measurement noise. Similarly, with the chain rule of differentiation, we can linearize (13.47) 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:

$$\begin{bmatrix} \mathbf{n}_{\pi}^c \\ d_{\pi}^c \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^c & \mathbf{0}_{3 \times 1} \\ -(\mathbf{p}_c^w)^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{n}_{\pi}^w \\ d_{\pi}^w \end{bmatrix} \quad (13.48)$$

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 closest point  $\mathbf{p}_{\pi}^c = d_{\pi}^c \mathbf{n}_{\pi}^c$  from the plane to the origin as the plane representation in the AINS state vector [255].

$$\mathbf{z}_{\pi} = d_{\pi}^c \mathbf{n}_{\pi}^c + \boldsymbol{\eta}^{\pi} = \mathbf{p}_{\pi}^c + \boldsymbol{\eta}^{\pi} \quad (13.49)$$

where  $\boldsymbol{\eta}^{\pi}$  is the plane measurement noise. Linearization of (13.49) yields the plane measurement Jacobian  $\mathbf{H}_x = \frac{\partial \mathbf{z}_{\pi}}{\partial \mathbf{p}_{\pi}^c} \frac{\partial \mathbf{p}_{\pi}^c}{\partial \mathbf{x}}$ .

### 13.3.2 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.* [326]):

$$\mathbf{M}(\hat{\mathbf{x}}) = \begin{bmatrix} \mathbf{H}_{x_1} \Phi_{(1,1)} \\ \mathbf{H}_{x_2} \Phi_{(2,1)} \\ \vdots \\ \mathbf{H}_{x_k} \Phi_{(k,1)} \end{bmatrix} \quad (13.50)$$

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  $\text{span}([\cdots \mathbf{u}_i \cdots]) = \mathcal{U}$  such that  $\mathbf{M}(\hat{\mathbf{x}})\mathbf{u}_i = \mathbf{0}$ , describes the unobservable subspace of AINS. If the nullspace is empty, the system is fully observable. It has been shown in [793] 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.* (13.42), (13.47), and (13.49)). By computing the related system and measurement Jacobians (*i.e.*,  $\mathbf{H}_{x_i}$  and  $\Phi_{(i,1)}$ ) and substituting them into (13.50), 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.* [793]):

$$\text{null}(\mathbf{M}) = \text{span}[\mathbf{u}_1 \ \mathbf{u}_{2:4}] = \text{span} \left[ \begin{array}{cc} \mathbf{u}_g & \mathbf{0}_{12 \times 3} \\ -\mathbf{p}_1^W \times \mathbf{g}^W & \mathbf{I}_3 \\ -\mathbf{p}_f^W \times \mathbf{g}^W & \mathbf{I}_3 \\ -\mathbf{g}^W & \frac{\mathbf{v}_\ell^W}{d_\ell^W \|\mathbf{v}_\ell^W\|} (\mathbf{R}_\ell^W \mathbf{e}_1)^T \\ 0 & -(\mathbf{R}_\ell^W \mathbf{e}_3)^T \\ -d_\pi^W \mathbf{n}_\pi^W \times \mathbf{g}^W & \mathbf{n}_\pi^W (\mathbf{R}_\pi^W \mathbf{e}_3)^T \end{array} \right] \quad (13.51)$$

where  $\mathbf{u}_g = [(\mathbf{R}_W^{C_1} \mathbf{g}^W)^T \ \mathbf{0}_{1 \times 3} \ -( \mathbf{v}_1^W \times \mathbf{g}^W )^T \ \mathbf{0}_{1 \times 3}]^T$ ,  $\mathbf{p}_1^W$  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.* (13.6)), and  $\mathbf{R}_\ell^W = \left[ \begin{array}{ccc} \mathbf{n}_\ell^W & \mathbf{v}_\ell^W & \mathbf{n}_\ell^W \\ \|\mathbf{n}_\ell^W\| & \|\mathbf{v}_\ell^W\| & \|\mathbf{n}_\ell^W\| \times \frac{\mathbf{v}_\ell^W}{\|\mathbf{v}_\ell^W\|} \end{array} \right]$  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 [793, 792] for more detailed analysis.

In summary, the fact that the observability matrix admits a 4-dimensional null space (*cf.* (13.51)) 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.

### 13.3.3 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 13.1 (see [793] 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 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.*,  $\|\mathbf{p}_f^C\| >> \|\mathbf{p}_B^C\|$ , which typically is the case in practice.

<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.

Table 13.1 *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

### 13.4 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 13.3.3). 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.

#### 13.4.1 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 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 [6], 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.

<sup>10</sup> In robotics, it is also common to use inertial data in combination with other sensors, including LiDAR and radars. We postpone the discussion about these other types of inertial odometry systems to Chapters 10 and 11.

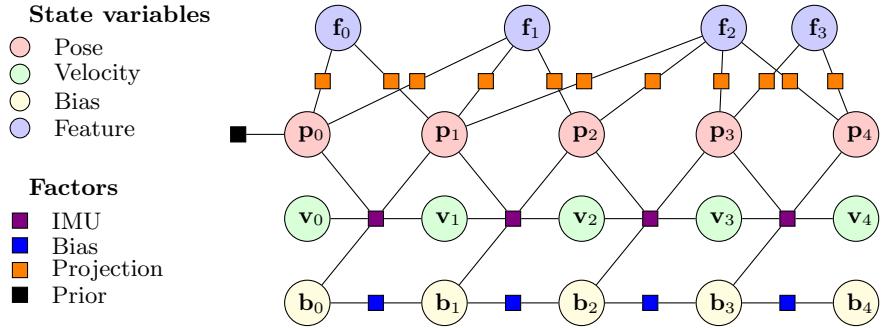


Figure 13.4 Example of factor graph used for visual-inertial odometry with preintegrated IMU factors [227]. The factor graph shows preintegrated IMU factors in violet (constraining consecutive poses, velocity, and bias), bias factors in blue (constraining the evolution of the IMU biases over time), vision factors in orange (relating camera poses and positions of external landmarks), and priors in black.

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. 13.4, 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.*, [227]. An alternative to using a fixed-lag smoother is to use an incremental solver like iSAM2 (Section 2.7), 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 [227], 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.

**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 [514], Direct Sparse Visual-Inertial Odometry [718], VINS-Mono [582], OpenVINS [256], Kimera [8, 607], BASALT [720], and DM-VIO [672]. A good VIO system has a 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 [581] 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 Win-

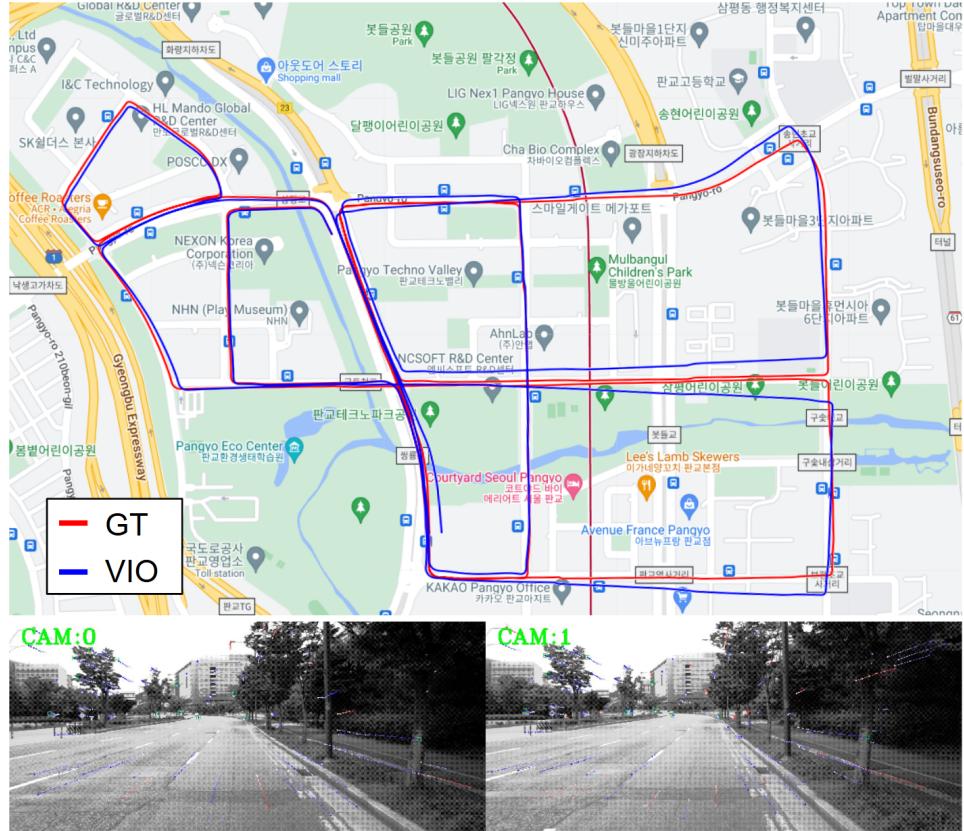


Figure 13.5 Illustration of a recent sliding-window optimization-based VIO algorithm [119] running on the KAIST urban autonomous driving dataset, sequence 38. This sequence has a total duration of 36 minutes and is 11.42 km in length. The VIO estimates and ground truth are overlaid on the Google map. 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%).

dow Bundle Adjustment (WBA)-VINS [119, 120] performs on the KAIST Urban Dataset [343]. 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 operates 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. 13.5 illustrates the FEJ-WBA-VINS [119, 120] (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 [8, 9], 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).

#### **13.4.2 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 [236], a known motion pattern [454], or a prior knowledge about the environment [415, 471]. 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 [199, 417, 775, 796]. 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 [794].

#### **13.4.3 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 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 [199, 257, 796].

### 13.5 New 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 [78] to improve the uncertainty modeling of IMU preintegration. The authors of [78] extend the preintegration theory to account for Earth's rotation with the Coriolis and centrifugal forces. The work [724] provides an example of extended pose preintegration leveraging both a linear velocity sensor and a navigational-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 [235] using B-splines basis functions and in [35] 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 [85] 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 [439], where the authors compare the GP-based state representation from [35] with the continuous GP-based preintegration from [257] (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 [296] with legged robots and [536] 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 [296] the critical information is the knowledge of contact between the robot’s feet and the ground, while in [536] the one-plane-rotation motion is used to constrain the IMU biases and therefore limit the dead-reckoning drift. The work [536] has been extended into a full SLAM system [768] 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 [172] 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 [777, 118, 678, 301, 302, 144, 584]. These include attempts to model IMU bias in a data-driven manner with neural networks [147] or directly predicting displacements from a sequence of noisy IMU measurements [450]. For instance, one may use a differentiable integration module to integrate IMU readings with the predicted bias removed [822, 584], or directly use ground truth bias for supervision [80], or use a conditional diffusion model to approximate bias which is modeled as a probability distribution [837]. 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 [5]. To reduce the data transfer, an on-sensor computing architecture is presented [266], and a quantized visual-inertial odometry (QVIO) algorithms is developed in [566, 568]. 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 [567, 763] have been introduced to improve effi-

ciency while maintaining numerical stability. An ASIC design and implementation for on-chip visual-inertial odometry system is presented in [823, 677].

# 14

## 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 [638]. This enables them to negotiate staircases, uneven terrain, and other ground obstacles that are challenging for wheeled platforms [143, 334]. 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 introduce the main fundaments to estimate 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 will introduce the theory to estimate leg odometry in Section 14.2 and Section 14.3, and how to combine it with different sensor modalities within factor graphs in Section 14.4. We will conclude with a review of the open problems in the field (Section 14.6), as well as new trends that have arisen to address them (Section 14.5).

### 14.1 Introduction

While most of modern legged platforms carry sensors we have studied in other chapters, namely cameras (Chapter 9), LiDAR (Chapter 10), and inertial measurement units (Chapter 13), in legged platforms we can additionally exploit kinematic 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 travelled 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.

In the next sections we will provide the technical details to implement leg odometry. In Section 14.2 we will explain how the relative motion can be obtained from the joint sensing of the legs, assuming that the stance legs are known. This practically allows us to consider leg odometry as a *measurement* in our SLAM or general estimation problems, in a similar way to wheel odometry or inertial preintegration. Then, Section 14.3 will describe how contact is estimated, and the main techniques adopted in practice.

#### 14.1.1 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 [446]. General Electric’s *Walking Truck* [507], 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 [492, 493, 703]. Raibert’s monopods, bipeds, and quadrupeds developed in the late 1980s showed remarkable locomotion capabilities [586], 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 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 [611]. Similar approaches were developed for other hexapods with different leg morphologies, such as RHex [444]. Later works combined leg odometry with other sensor modalities such as IMU and vision, via particle filters [136] or Kalman filters [130, 600]. The milestone work by Bloesch *et al.* [62] established the foundations for filtering-based, all terrain, kinematic-inertial odometry estimators for quadrupeds [63], and extensions to bipedal platforms [612, 574].

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 but also the full state of the robot’s body. For this, various teams combined kinematic and dynamic information [770] with

inertial and joint sensing, as well as exteroceptive modalities such as LiDAR [212] and stereo vision [213]. In this period most of the solutions relied on Kalman filtering but a few works also explored optimization-based solutions [771] and factor graphs [228].

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 [762] and bipedal platforms [294], which have enabled its principled fusion with other sensor modalities. Other directions have been explored more fundamental challenges in modelling [21], as well as invariant estimation frameworks [297, 293]. 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 [371, 830, 551]. Section 14.6 will provide further insights on the recent trends, and how they are re-shaping the research in state estimation for legged platforms.

#### 14.1.2 Reference Frames

In Figure 14.1, we illustrate the reference frames relevant to our estimation problem. The inertial frame  $\mathbf{W}$  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 to  $\mathcal{F}^b$ . 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 14.1).

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.

#### 14.1.3 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 14.1.6), 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 [91].

More formally, the robot state is defined as the set combining position, orientation, linear velocity, and angular velocity:

$$\boldsymbol{x}_k = [\boldsymbol{p}_k \quad \boldsymbol{R}_k \quad \boldsymbol{v}_k \quad \boldsymbol{\omega}_k]^T \quad (14.1)$$

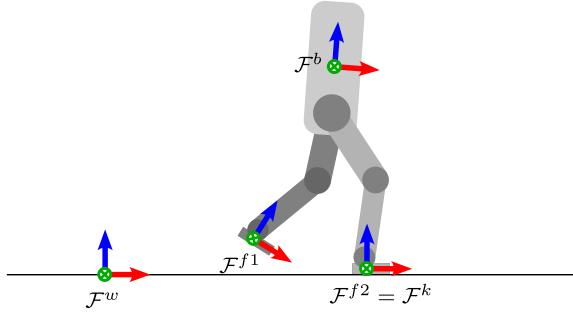


Figure 14.1 Reference frame conventions for legged robots. The world frame  $\mathbf{W}$  is fixed to earth, while the base frame  $\mathcal{F}^b$  is attached to the main chassis. Without loss of generality, the IMU frame  $B$  is not shown as it can be considered coincident with  $\mathcal{F}^b$ . 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  $\mathcal{F}^f$ .

where the following conventions are adopted: the robot position  $\mathbf{p} = \mathbf{t}_b^w \in \mathbb{R}^3$  and orientation  $\mathbf{R} = \mathbf{R}_b^w \in \text{SO}(3)$  express the pose of the robot's base in world coordinates; the robot velocities  $\mathbf{v} = \mathbf{v}_b^b$ ,  $\boldsymbol{\omega} = \boldsymbol{\omega}_b^b \in \mathbb{R}^3$  express the base's twist, in base coordinates.

#### 14.1.4 Legged Robot Kinematics

A legged robot is kinematically described by a main link for the body (also referred as 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 14.2). 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  $\mathbf{q} \in \mathbb{R}^N = [q_1, \dots, q_N]^\top$  be the set of joint positions of an articulated robot with  $N$  active DoF, which correspond to the angular position of revolute joints of the legs. In Figure 14.1 and for the rest of the chapter, the active DoFs of the quadruped and biped is  $N = 12$ , although this can be different in other platforms. The joint positions  $\mathbf{q}$  are typically measured directly via rotary encoders placed on each joint (see Section 14.1.6.1), or indirectly using the kinematic model of the robot in addition to the readings from the encoders placed on a transmission between the 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  $\dot{\mathbf{q}} = [\dot{q}_1, \dots, \dot{q}_N]^\top$ ,

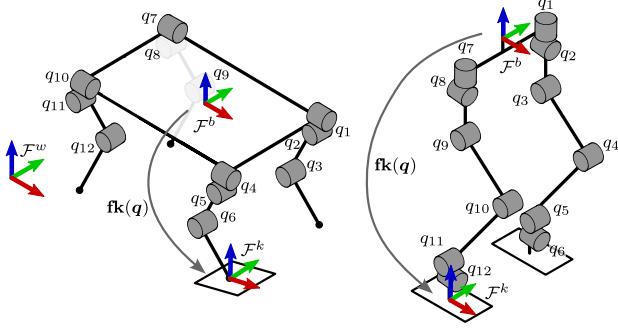


Figure 14.2 Kinematic chains of typical legged robots: quadrupeds and humanoids.

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 [772].

Given  $\mathbf{q}$  and a specific foot  $f$  of a humanoid robot, the *Forward kinematics* function  $\mathbf{fk}(\mathbf{q}) : \mathbb{R}^{12} \rightarrow \text{SE}(3)$  [474] maps the joint positions to the pose of the foot respect to the robot's base:

$$\mathbf{T}_f^b = \mathbf{fk}(\mathbf{q}) = \begin{bmatrix} \mathbf{f}_R(\mathbf{q}) & \mathbf{f}_p(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix} \quad (14.2)$$

where we defined 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 from (14.2) 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 the end effector respect to the robot's base as follows [475]:

$$\begin{bmatrix} \mathbf{v}_f^b \\ \boldsymbol{\omega}_f^b \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} \mathbf{J}_v(\mathbf{q}) \\ \mathbf{J}_\omega(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} \quad (14.3)$$

where  $\mathbf{J}_v(\mathbf{q})$  and  $\mathbf{J}_\omega(\mathbf{q})$  are the linear and angular part of the Jacobian respectively (note that some references [216][297] define the angular block first). Since each leg is kinematically independent from the other,  $\mathbf{J}(\mathbf{q})$  is as a sparse block matrix, where the only two non-zero blocks  $\mathbf{J}_{f,v}(\mathbf{q})$ ,  $\mathbf{J}_{f,\omega}(\mathbf{q})$  map the subset of joint angle velocities of a leg to the linear and angular velocity of the corresponding foot  $f$ . For example, the Jacobian of the second leg of a humanoid  $\mathbf{J}_2(\mathbf{q})$  is represented as:

$$\mathbf{J}_2(\mathbf{q}) = [\mathbf{0}_6 \quad \bar{\mathbf{J}}_2(\mathbf{q})] \quad (14.4)$$

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 (14.2) to (14.3) 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 14.1.6.

#### 14.1.5 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 [216]. The first equation describes the dynamics of the floating-base body (6 DoF, underactuated), 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:

$$\boldsymbol{M}(\boldsymbol{q}) \begin{bmatrix} \dot{\boldsymbol{v}}_b^b \\ \dot{\boldsymbol{\omega}}_b^b \\ \ddot{\boldsymbol{q}} \end{bmatrix} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \mathbf{J}_b^\top \\ \mathbf{J}_q^\top \end{bmatrix} \boldsymbol{f} + \begin{bmatrix} \mathbf{0}_6 \\ \boldsymbol{\tau} \end{bmatrix} \quad (14.5)$$

where the first term  $\boldsymbol{M}(\boldsymbol{q})$  is the mass matrix, which is multiplied by the stack of  $\dot{\boldsymbol{v}} \in \mathbb{R}^3$ ,  $\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$ , and  $\ddot{\boldsymbol{q}} \in \mathbb{R}^{12}$  which represent the floating-base linear, floating-base angular, and active joints accelerations, respectively. The second term  $\boldsymbol{h} \in \mathbb{R}^{18}$  is a bias term that accounts for Coriolis, centrifugal, and gravitational effects. Regarding the right side of (14.5), 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 W.  $\mathbf{J}_q \in \mathbb{R}^{dc \times 12}$  is the stack of Jacobians described in (14.3) whose arrangement depends on the type of robot and number of contact legs. For example, a humanoid standing on both legs will have:

$$\mathbf{J}_q = \begin{bmatrix} \mathbf{J}_1(\boldsymbol{q}) \\ \mathbf{J}_2(\boldsymbol{q}) \end{bmatrix} \quad (14.6)$$

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:

$$\mathbf{J}_q = \begin{bmatrix} \mathbf{J}_{1,v}(\boldsymbol{q}) \\ \mathbf{J}_{3,v}(\boldsymbol{q}) \\ \mathbf{J}_{4,v}(\boldsymbol{q}) \end{bmatrix} \quad (14.7)$$

The last term to consider is  $\boldsymbol{f} \in \mathbb{R}^{dc}$  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:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \mathbf{f}_4 \end{bmatrix} \quad (14.8)$$

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:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \boldsymbol{\tau}_1 \\ \mathbf{f}_2 \\ \boldsymbol{\tau}_2 \end{bmatrix} \quad (14.9)$$

As explained in more detail in Section 14.3.4, we can infer leg contact using the aforementioned forces and torques.

#### 14.1.6 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 [654]. We briefly describe the most important sensors that are used also for leg odometry, namely encoders, force/torque sensors, and contact sensors.

##### 14.1.6.1 Rotary Encoders

Rotary encoders are electromechanical 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 LiDAR use them to measure the azimuthal angle of the beam array (see Chapter 10).

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 [476].

Absolute optical digital encoders (Figure 14.3a) 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

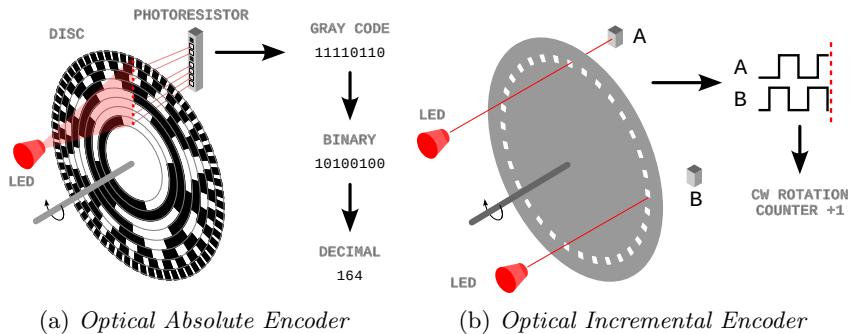


Figure 14.3 Left: Principle of operation of an 8-bit optical absolute encoder. An IR light beam hits a rotating disc that masks light according to a specific pattern that encodes the angle of rotation of the disc. An array of photoresistors convert the absence or presence of the light in each sector 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. Right: 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.

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 14.3a 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 optical encoders (Figure 14.3b), conversely, measure relative angular changes with respect to the *initial configuration*—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 [476]. 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 [637]. 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.

#### 14.1.6.2 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 14.4 shows an example with strain gauges applied to a load cell to measure linear force [476]. To measure torque, a series of strain gauges is applied to flexible spokes of a wheel connected to a motor shaft.

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 advent of cost effective dynamic legged robots, which was made possible by a backdriveable motor design [359], torques can be estimated from the motor currents, which are proportional to the torque by a constant factor.

#### 14.1.6.3 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 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 [267] or mechanical [525]. 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

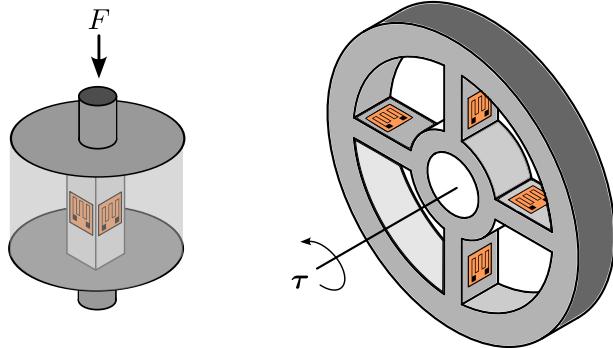


Figure 14.4 Principle of operation of force and torque sensors. A strain gauge is applied on compressible or flexible elements when under load. Inside the loadcell (on the left), a strain gauge is applied such that a compression would be detected as a reduction in resistivity. Inside the torque sensor (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.

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.

## 14.2 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 instantaneously. In both cases, the underlying assumption is that a newly formed contact frame remains stationary for a certain amount of time.

### 14.2.1 Relative Pose Estimation

Figure 14.5 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

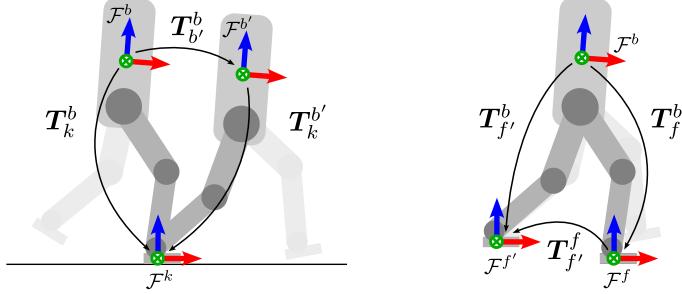


Figure 14.5 How leg odometry works with ideal contact. Left: Assuming the contact frame  $\mathcal{F}^k$  is rigidly attached to the ground (in yellow), we can determine the relative motion of the robot’s body. Right: Alternatively, we can represent how the leg moves with respect to the body frame (yellow) in two consecutive instants.

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:

$$\mathbf{T}_{b'}^b = \mathbf{T}_k^b (\mathbf{T}_k^{b'})^{-1} = (\mathbf{T}_{f'}^b)^{-1} = (\mathbf{T}_{f'}^b)^{-1} \mathbf{T}_f^b = \mathbf{fk}(\mathbf{q}')^{-1} \mathbf{fk}(\mathbf{q}) \quad (14.10)$$

(14.10) 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 [611], 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 [62] is to augment the state in (14.1) 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 [612].

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 (14.1) is disregarded, and the IMU biases are included as part of the state instead (see Chapter IMU).

With all the previous considerations, the corresponding states of interest for quadrupeds and bipeds are then defined as:

$$\mathbf{x}_k = [\mathbf{p}_k \quad \mathbf{R}_k \quad \mathbf{v}_k \quad \mathbf{b}_k^a \quad \mathbf{b}_k^\omega \quad \mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{c}_3 \quad \mathbf{c}_4]^\top \quad (14.11)$$

$$\mathbf{x}_k = [\mathbf{p}_k \quad \mathbf{R}_k \quad \mathbf{v}_k \quad \mathbf{b}_k^a \quad \mathbf{b}_k^\omega \quad \mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{B}_1 \quad \mathbf{B}_2]^\top \quad (14.12)$$

where (14.11) represents the state of a quadruped robot, whilst (14.12) corresponds to a humanoid robot. This enables us to precisely express the motion estimate relationship from (14.10) for an arbitrary  $i$ -th leg:

$$\mathbf{T}_k^b = \mathbf{fk}(\mathbf{q}) = (\mathbf{T})^{-1}\mathbf{C}_i \quad (14.13)$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \quad (14.14)$$

is the pose of the base in the fixed frame, whereas

$$\mathbf{C}_i = \begin{bmatrix} \mathbf{B}_i & \mathbf{c}_i \\ \mathbf{0} & 1 \end{bmatrix} \quad (14.15)$$

is the pose of the foot contact in the fixed frame. Expanding (14.13) leads to:

$$(\mathbf{T})^{-1}\mathbf{C}_i = \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{B}_i & \mathbf{c}_i \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^\top \mathbf{B}_i & \mathbf{R}^\top \mathbf{c}_i - \mathbf{R}^\top \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \quad (14.16)$$

Rearranging (14.16), we can define the following components corresponding to the upper blocks of right-hand side matrix:

$$\mathbf{f}_p(\mathbf{q}) = \mathbf{R}^\top(\mathbf{c}_i - \mathbf{p}) \quad (14.17)$$

$$\mathbf{f}_R(\mathbf{q}) = \mathbf{R}^\top \mathbf{B}_i \quad (14.18)$$

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 (14.17), since we cannot obtain an orientation estimate from point feet.

(14.17) and (14.18) are the basic leg odometry expressions used as measurements within estimation frameworks such as filters or factor graphs. These will be further described in Section 14.4.

### 14.2.2 Velocity Estimation

The differential kinematics function from (14.3) can be used get a direct velocity measurement from each stance leg. This approach is widely adopted on quadrupeds [63, 97, 387] and less frequently on bipeds [695]. 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 [212] 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:

$$\mathbf{v}_k^w = \mathbf{0}. \quad (14.19)$$

Furthermore, the velocity of the contact point  $k$  must coincide with the velocity of the foot  $f$ , also described by the Jacobian matrix (14.3):

$$\mathbf{v}_k^b = \mathbf{v}_f^b = \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}}. \quad (14.20)$$

Since the angular velocity of the robot is measured by the IMU, we focus on the linear velocity only. From (14.19) and (14.20), we can determine the robot's body velocity as:

$$\begin{aligned} \mathbf{v}_k^w &= \mathbf{v}_b^w + \boldsymbol{\omega}_b^w \times \mathbf{t}_k^b + \mathbf{v}_k^b \\ \mathbf{0} &= \mathbf{v}_b^w + \boldsymbol{\omega}_b^w \times \mathbf{f}_p(\mathbf{q}) + \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} \\ \mathbf{v}_b^w &= -\boldsymbol{\omega}_b^w \times \mathbf{f}_p(\mathbf{q}) - \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} \end{aligned} \quad (14.21)$$

where we take into account the additional linear velocity caused by the lever arm between the base and the foot [175]. (14.21) 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 14.4. 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.

### 14.3 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.

Technically, a foot does not slip when the vertical component of the Ground Reaction Force (GRF),  $f_z$ , is within the friction cone [613]:

$$\sqrt{f_x^2 + f_y^2} \leq \mu_{x,y} f_z \quad (14.22)$$

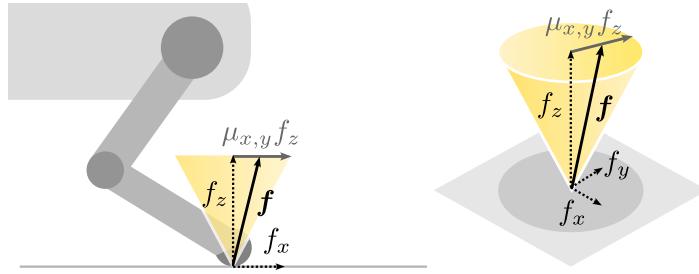


Figure 14.6 The contact point on a quadruped’s leg. A leg can be considered in stance when the force  $\mathbf{f} = [f_x, f_y, f_z]^\top$  applied by the foot stays within the friction cone.

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 (14.22), which requires that the foot does not rotate:

$$\begin{bmatrix} -\tau_y/f_z \\ \tau_x/f_z \end{bmatrix} \leq \begin{bmatrix} CoP_x \\ CoP_y \end{bmatrix} \quad (14.23)$$

$$|\tau_z| \leq \mu_z f_z \quad (14.24)$$

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 ((14.22)-(14.24)) 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.

#### 14.3.1 With Contact Sensors

Contact sensors implicitly threshold  $f_z$  in hardware, as they are tuned such that the binary signal they provide 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.

### 14.3.2 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 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 [212].

### 14.3.3 With IMUs

As seen in Chapter 13, IMUs are inexpensive sensors that provide acceleration and rotational velocity measurements. While we tipically 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 [790, 613, 482]. 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.

### 14.3.4 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's dynamics (see (14.1.5)).

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 (14.5) as:

$$\mathbf{f}_i = -(\bar{\mathbf{J}}_{i,v}^T)^{-1}(\boldsymbol{\tau}_i - \mathbf{h}_{q,i} - \mathbf{F}^T \dot{\mathbf{v}}) \quad (14.25)$$

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) [222]

- 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 [342].

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 [331]) or by using learning methods (see Section 14.6.1).

## 14.4 Leg Odometry for Estimation Problems

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.

### 14.4.1 Encoder Noise Propagation

The main source of uncertainties 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  $\boldsymbol{q}$  and the measured value  $\tilde{\boldsymbol{q}}$  are related as follows:

$$\tilde{\boldsymbol{q}} = \boldsymbol{q} + \boldsymbol{\eta}_q \quad (14.26)$$

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 [294]:

$$\mathbf{f}_p(\boldsymbol{q} + \boldsymbol{\eta}_q) \approx \mathbf{f}_p(\boldsymbol{q}) + \mathbf{J}_c(\boldsymbol{q})\boldsymbol{\eta}_q \quad (14.27)$$

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 Gaussianity assumption can also be applied to velocity measurements affected by encoder noise  $\boldsymbol{\eta}_q$  and encoder velocity noise  $\boldsymbol{\eta}_{\dot{q}}$  [762], considering that:

$$\mathbf{J}(\mathbf{q} + \boldsymbol{\eta}_q)(\dot{\mathbf{q}} + \boldsymbol{\eta}_{\dot{q}}) \approx \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} + \frac{\partial}{\partial \mathbf{q}} (\mathbf{J}(\mathbf{q})\dot{\mathbf{q}}) \boldsymbol{\eta}_q + \mathbf{J}(\mathbf{q})\boldsymbol{\eta}_{\dot{q}} \quad (14.28)$$

From (14.27) and (14.28), 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.

#### 14.4.2 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) [62, 97], the Unscented Kalman Filter (UKF) [63], or the Invariant-EKF [297, 811, 445]. 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 [98] 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) [65] 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 13.

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 [295], and velocity bias preintegration for quadrupeds [762]. In both cases, the measurements from Section 14.2 will be reformulated in terms of residuals and covariances for the factors of the graph.

#### 14.4.2.1 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 (14.12). This idea was presented by Hartley *et al.* [295], and the proposed factor graph is shown in Figure 14.7a.

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* 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 (14.27) into the relative pose measurement of (14.17) and (14.18), we can define the following residual and covariance for the forward kinematics factor:

$$\mathbf{r}_F = \text{Log}(\mathbf{C}_i^{-1} \mathbf{T} \mathbf{fk}(\tilde{\mathbf{q}})) \quad (14.29)$$

$$\boldsymbol{\Sigma}_F = \mathbf{J}_c(\tilde{\mathbf{q}}) \boldsymbol{\Sigma}_q \mathbf{J}_c^\top(\tilde{\mathbf{q}}) \quad (14.30)$$

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* 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:

$$\Delta \tilde{\mathbf{B}}_{ij} = \mathbf{B}_i^\top \mathbf{B}_j \text{Exp}(\delta \boldsymbol{\theta}_{ij}) = \mathbf{I} \quad (14.31)$$

$$\Delta \tilde{\mathbf{c}}_{ij} = \mathbf{B}_i^\top (\mathbf{c}_j - \mathbf{c}_i) + \delta \mathbf{d}_{ij} = \mathbf{0} \quad (14.32)$$

where we have used the rotational and translational components of the contact

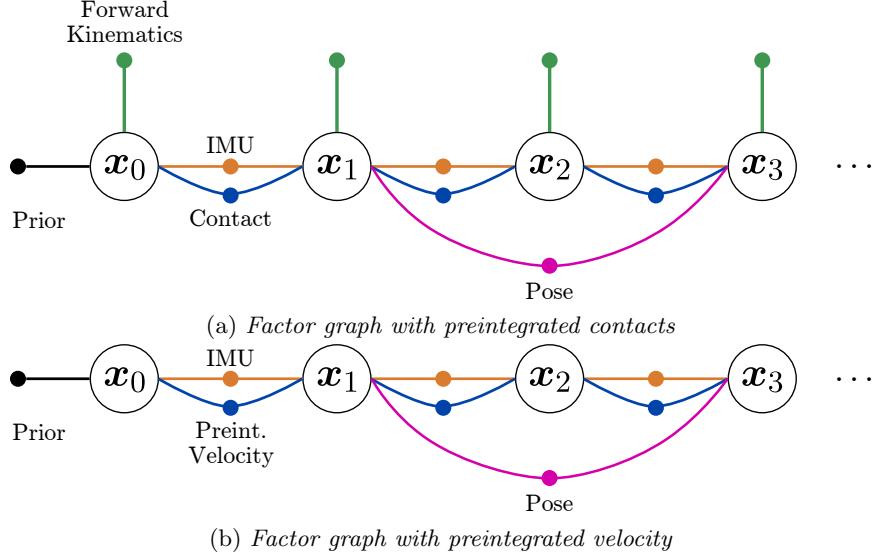


Figure 14.7 Factor graph formulations for preintegrated contact (top) and velocity (bottom). Additional measurements (*e.g.*, from exteroceptive sensors, which constrain two states) can be easily added as additional factors (magenta).

frames  $\mathbf{C}_i$  and  $\mathbf{C}_j$  as stated in (14.15). The terms  $\delta\boldsymbol{\theta}_{ij}$  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 [295]. The preintegrated contact factor is then given by the following rotation and translation residuals and covariance:

$$\mathbf{r}_C = \begin{bmatrix} \text{Log}(\mathbf{B}_i^\top \mathbf{B}_j) \\ \mathbf{B}_i^\top (\mathbf{c}_j - \mathbf{c}_i) \end{bmatrix} \quad (14.33)$$

$$\boldsymbol{\Sigma}_C = \begin{bmatrix} \boldsymbol{\Sigma}_w & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_v \end{bmatrix} \Delta t_{ij} \quad (14.34)$$

where we have rearranged and stacked (14.31) and (14.32) 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 [294], by modeling and properly handling the contact frame switches, enabling preintegration among different legs.

#### 14.4.2.2 Velocity Preintegration

As mentioned previously, the kinematics of point feet platforms—such as quadruped robots—cannot constrain the relative 6 DoF 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 14.2.2, and preintegrate them to obtain additional factors that constraint the relative change of the robot’s pose [762, 387].

*Velocity Preintegration Factor* This factor assumes that the instant linear velocity of the body can be determined from leg odometry using (14.28) and it is affected by Gaussian noise terms  $\boldsymbol{\eta}_v$  and  $\boldsymbol{\eta}_\omega$ :

$$\tilde{\mathbf{v}} = -\mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} - \boldsymbol{\omega} \times \mathbf{f}_p(\mathbf{q}) + \boldsymbol{\eta}_v \quad (14.35)$$

Assuming the robot has constant body linear velocity between times  $t_i$  and  $t_j$ , we can preintegrate the velocity measurements to obtain:

$$\Delta\tilde{\mathbf{p}}_{ij} = \Delta\mathbf{p}_{ij} + \delta\mathbf{p}_{ij} = \sum_{k=i}^{j-1} [\Delta\tilde{\mathbf{R}}_{ik}\tilde{\mathbf{v}}_k \Delta t] + \delta\mathbf{p}_{ij} \quad (14.36)$$

where, similarly to the preintegrated contact factors,  $\delta\mathbf{p}_{ij}$  is a *preintegrated velocity noise* term [760, 387]. Then, the preintegrated velocity factor and associated covariance are given by:

$$\mathbf{rV} = \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i) - \Delta\mathbf{p}_{ij} \quad (14.37)$$

$$\boldsymbol{\Sigma}_{V,ij} = \sum_{j=1}^{k=i} \boldsymbol{\Sigma}_{V,ik} + \mathbf{A}\boldsymbol{\Sigma}_v\mathbf{A}^\top \quad (14.38)$$

with the matrix  $\mathbf{A} = \Delta\tilde{\mathbf{R}}_{ik}\Delta t$ .

#### 14.4.2.3 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 [212] 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 [295, 387]. 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 [762].

#### **14.4.3 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 pose graph) 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 [196, 371].

The alternative to loosely coupled methods are tightly coupled ones. In [762] 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.

### **14.5 Open Challenges**

In previous sections we have covered how to generally perform leg odometry and fuse it with other sensor modalities. We made a number of assumptions that are often not valid in practice and challenging situations that are still unaddressed. We briefly introduce them here.

#### **14.5.1 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 14.8). 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-

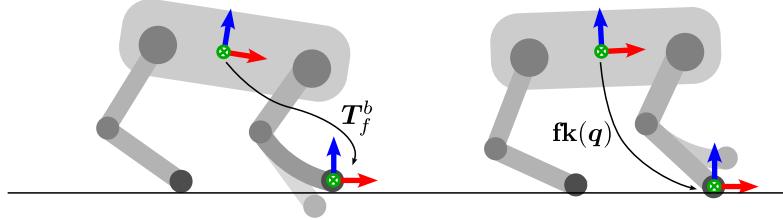


Figure 14.8 Example of leg deformation on a quadruped. The real transformation between the robot base and the contact point is shown at the left. Since the forward kinematics assumes the robot's legs are rigid, it incorrectly estimates an upward motion, as shown on the right.

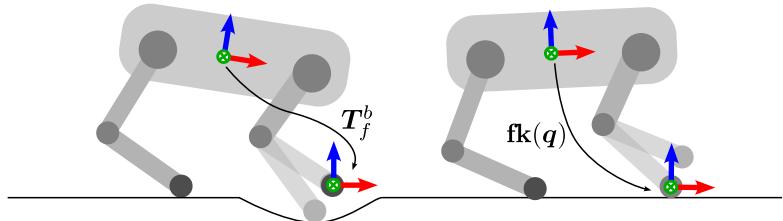


Figure 14.9 Example of ground deformation with a quadruped. The robot initially touches the ground which is flat. While keeping the contact state on, the ground deforms and the foot sinks into it (left). As the joint angles change while the foot goes down, this motion is interpreted as an upward motion 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 [97].

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 cannot be generalized 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 [725].

### 14.5.2 Non-rigid Contacts and Slippage

If the robot is walking on soft or collapsible ground, when the contact is first detected before the ground starts its deformation (Figure 14.9, left). Then, the leg stretches penetrating the terrain. Because the contact point is assumed to be stationary, this is interpreted by the forward kinematics as an upward motion (Figure 14.9, 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 [211], 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 (14.22) and/or (14.24).

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 [695]. The velocity of the contact can be explicitly tracked as an additional state in [762], or as the derivative of the feet positions [387].

## 14.6 New Trends and Paradigm Shifts

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.

### 14.6.1 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 [97] but also in an unsupervised fashion via clustering [612], 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 of to a wider set of structured and unstructured environments [445]. Vision-based haptic sensors, which capitalize on the progress of machine learning and computer vision [424], are another direction that shows promise to improve force and contact estimates [647].

### 14.6.2 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 locomotions controllers showed that only the body velocity and orientation are required for locomotion, which can be provided explicitly by a standard proprioceptive state estimators [332], or even raw data from joint and inertial sensing [422, 499].

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 IMU, 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 [846, 313]. 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 [344], 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.

#### 14.6.3 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 14.1.1, 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 un-

explored. 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 relaxes the rigid contact assumptions we made in this chapter.

### **Acknowledgment**

The authors thank Michele Focchi (University of Trento) for his advice in preparing parts of this chapter.

## **PART THREE**

### FROM SLAM TO SPATIAL AI



# 15

## Prelude

Author I, Author II, and Author III

Aenean sem dolor, fermentum nec, gravida hendrerit, mattis eget, felis. Nullam non diam vitae mi lacinia consectetur. Fusce non massa eget quam luctus posuere. Aenean vulputate velit. Quisque et dolor. Donec ipsum tortor, rutrum quis, mollis eu, mollis a, pede. Donec nulla. Duis molestie. Duis lobortis commodo purus. Pellentesque vel quam. Ut congue congue risus. Sed ligula. Aenean dictum pede vitae felis. Donec sit amet nibh. Maecenas eu orci. Quisque gravida quam sed massa.

[162]

# **16**

## Spatial AI

Author I, Author II, and Author III

Aenean sem dolor, fermentum nec, gravida hendrerit, mattis eget, felis. Nullam non diam vitae mi lacinia consectetur. Fusce non massa eget quam luctus posuere. Aenean vulputate velit. Quisque et dolor. Donec ipsum tortor, rutrum quis, mollis eu, mollis a, pede. Donec nulla. Duis molestie. Duis lobortis commodo purus. Pellentesque vel quam. Ut congue congue risus. Sed ligula. Aenean dictum pede vitae felis. Donec sit amet nibh. Maecenas eu orci. Quisque gravida quam sed massa.

# 17

## Boosting SLAM with Deep Learning

Author I, Author II, and Author III

Aenean sem dolor, fermentum nec, gravida hendrerit, mattis eget, felis. Nullam non diam vitae mi lacinia consectetur. Fusce non massa eget quam luctus posuere. Aenean vulputate velit. Quisque et dolor. Donec ipsum tortor, rutrum quis, mollis eu, mollis a, pede. Donec nulla. Duis molestie. Duis lobortis commodo purus. Pellentesque vel quam. Ut congue congue risus. Sed ligula. Aenean dictum pede vitae felis. Donec sit amet nibh. Maecenas eu orci. Quisque gravida quam sed massa.

# 18

## NeRF and 3D Gaussian Splatting: Map Representations with Differentiable Volume Rendering

Author I, Author II, and Author III

Aenean sem dolor, fermentum nec, gravida hendrerit, mattis eget, felis. Nullam non diam vitae mi lacinia consectetur. Fusce non massa eget quam luctus posuere. Aenean vulputate velit. Quisque et dolor. Donec ipsum tortor, rutrum quis, mollis eu, mollis a, pede. Donec nulla. Duis molestie. Duis lobortis commodo purus. Pellentesque vel quam. Ut congue congue risus. Sed ligula. Aenean dictum pede vitae felis. Donec sit amet nibh. Maecenas eu orci. Quisque gravida quam sed massa.

# 19

## Dynamic and Deformable SLAM

Author I, Author II, and Author III

Aenean sem dolor, fermentum nec, gravida hendrerit, mattis eget, felis. Nullam non diam vitae mi lacinia consectetur. Fusce non massa eget quam luctus posuere. Aenean vulputate velit. Quisque et dolor. Donec ipsum tortor, rutrum quis, mollis eu, mollis a, pede. Donec nulla. Duis molestie. Duis lobortis commodo purus. Pellentesque vel quam. Ut congue congue risus. Sed ligula. Aenean dictum pede vitae felis. Donec sit amet nibh. Maecenas eu orci. Quisque gravida quam sed massa.

# **20**

## Metric-Semantic SLAM

Author I, Author II, and Author III

# **21**

## Foundation Models for SLAM

Author I, Author II, and Author III



## Notes

## References

- [1] *FastTriggs*. <https://pypose.org/docs/main/generated/pypose.optim.corrector.FastTriggs/>.
- [2] *LieTensor*. <https://pypose.org/docs/main/generated/pypose.LieTensor>.
- [3] *PyPose Documents*. <https://pypose.org/docs/>.
- [4] *PyPose Tutorial*. <https://github.com/pypose/tutorials>.
- [5] 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/>.
- [6] 2025 (Mar.). *Meta Quest 3: Technical Specifications*.
- [7] 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)*.
- [8] 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)*.
- [9] 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)*. ,.
- [10] 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)*.
- [11] Adolfsson, Daniel, Castellano-Quero, Manuel, Magnusson, Martin, Lilienthal, Achim J., and Andreasson, Henrik. 2022. CorAI: Introspection for robust radar and lidar perception in diverse environments using differential entropy. *Robotics and Autonomous Systems*, May, 104136.
- [12] 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.
- [13] 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.
- [14] Aftab, Khurrum, 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.
- [15] 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)*.
- [16] 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)*.
- [17] 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)*.
- [18] 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.
- [19] Agarwal, Sameer, Mierle, Keir, and Team, The Ceres Solver. 2022 (3). *Ceres Solver*.
- [20] 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*.
- [21] 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*.
- [22] 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 e-prints*, arXiv–1605.
- [23] 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).
- [24] 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)*.
- [25] 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.
- [26] 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.
- [27] 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)*.

- [28] A.N. Mopidevi, K. Harlow, C. Heckman. 2024 (Oct.). RMap: Millimeter-Wave Radar Mapping Through Volumetric Upsampling. In: *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [29] 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.
- [30] Ankenbauer, Jacqueline, Lusk, Parker C., and How, Jonathan P. 2023 (Mar.). *Global Localization in Unstructured Environments Using Semantic Object Maps Built from Various Viewpoints*.
- [31] 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. .
- [32] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., and Sivic, J. 2016a. NetVLAD: CNN architecture for weakly supervised place recognition. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [33] Arandjelovic, Relja, Gronat, Petr, Torii, Akihiko, Pajdla, Tomas, and Sivic, Josef. 2016b. NetVLAD: CNN architecture for weakly supervised place recognition. Pages 5297–5307 of: *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- [34] 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)*.
- [35] Barfoot, T D. 2024. *State Estimation for Robotics*. 2nd edn. Cambridge University Press.
- [36] 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.
- [37] Barfoot, T D, Forbes, J R, and D'Eleuterio, G M T. 2022. Vectorial Parameterizations of Pose. *Robotica*, **40**(7), 2409–2427.
- [38] 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.
- [39] Barfoot, Timothy D. 2017. *State estimation for robotics*. Cambridge University Press.
- [40] 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)*.
- [41] Barnes, Dan, Weston, Rob, and Posner, Ingmar. 2019. Masking by Moving: Learning Distraction-Free Radar Odometry from Pose Information. In: *Conference on Robot Learning*.
- [42] 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)*.
- [43] Barranco, Francisco, Fermuller, Cornelia, Aloimonos, Yiannis, and Delbruck, Tobi. 2016. A Dataset for Visual Navigation with Neuromorphic Methods. *Front. Neurosci.*, **10**, 49.

- [44] Barrau, A., and Bonnabel, S. 2017. The Invariant Extended Kalman Filter as a Stable Observer. *IEEE Trans. on Automatic Control*, **62**(4), 1797–1812.
- [45] Barron, Jonathan T. 2019. A general and adaptive robust loss function. Pages 4331–4339 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [46] Bauernfeind, Carl Max v. 1856. *Elemente der Vermessungskunde: ein Lehrbuch der praktischen Geometrie*. Cotta.
- [47] Bay, Herbert, Tuytelaars, Tinne, and Van Gool, Luc. 2006. SURF: Speeded Up Robust Features. In: *European Conf. on Computer Vision (ECCV)*. Springer Berlin Heidelberg.
- [48] 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.
- [49] 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)*.
- [50] Behley, J., and Stachniss, C. 2018. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In: *Robotics: Science and Systems (RSS)*.
- [51] 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)*.
- [52] 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)*.
- [53] Bentley, J.L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, **18**(9), 509–517.
- [54] 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.
- [55] 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.
- [56] Bezdek, James C., and Hathaway, Richard J. 2003. Convergence of alternating optimization. *Neural, Parallel & Scientific Computations*, **11**(4), 351–368.
- [57] 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.
- [58] 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.
- [59] Bierman, G.J. 1977. *Factorization methods for discrete sequential estimation*. Mathematics in Science and Engineering, vol. 128. New York: Academic Press.
- [60] 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.
- [61] Blake, Andrew, and Zisserman, Andrew. 1987. *Visual reconstruction*. MIT Press.

- [62] 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)*.
- [63] 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)*.
- [64] 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)*.
- [65] Bloesch, Michael, Burri, Michael, Sommer, Hannes, Siegwart, Roland, and Hutter, Marco. 2018. The Two-State Implicit Filter Recursive Estimation for Mobile Robots. *IEEE Robotics and Automation Letters*, **3**(1), 573–580.
- [66] 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.
- [67] Bosse, M., Agamennoni, G., and Gilitschenski, I. 2016. Robust Estimation and Applications in Robotics. *Foundations and Trends in Robotics*, **4**(4), 225–269.
- [68] 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)*.
- [69] 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.
- [70] 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.
- [71] Botsch, M., Spernats, M., and Kobbelt, L. 2004. Phong splatting. In: *Symposium on Point-Based Graphics (PBG)*.
- [72] 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)*.
- [73] Boumal, Nicolas. 2022 (Mar). *An introduction to optimization on smooth manifolds*. To appear with Cambridge University Press.
- [74] Boyle, Michael. 2017. The Integration of Angular Velocity. *Advances in Applied Clifford Algebras*, **27**(3), 2345–2374.
- [75] Bradbury, James, Frostig, Roy, Hawkins, Peter, Johnson, Matthew James, Leary, Chris, Maclaurin, Dougal, Necula, George, Paszke, Adam, VanderPlas, Jake, Wanderman-Milne, Skye, and Zhang, Qiao. 2018. *JAX: composable transformations of Python+NumPy programs*.
- [76] 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.

- [77] Brogan, William L. 1991. *Modern Control Theory*. Upper Saddle River, NJ: Prentice Hall.
- [78] 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.
- [79] 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).
- [80] 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.
- [81] Burner, Levi, Mitrokhin, Anton, Fermüller, Cornelia, and Aloimonos, Yiannis. 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.
- [82] 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.
- [83] 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.
- [84] Burnett, Keenan, Schoellig, Angela P., and Barfoot, Timothy D. 2024. *Continuous-Time Radar-Inertial and Lidar-Inertial Odometry using a Gaussian Process Motion Prior*.
- [85] 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.
- [86] 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. *The International Journal of Robotics Research*.
- [87] 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.
- [88] 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 arXiv:1902.01534*.
- [89] 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.
- [90] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. 2016a. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Trans. Robotics*, **32**(6), 1309–1332.

- [91] Cadena, Cesar, Carlone, Luca, Carrillo, Henry, Latif, Yasir, Scaramuzza, Davide, Neira, José, Reid, Ian, and Leonard, John J. 2016b. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, **32**(6), 1309–1332.
- [92] Caesar, Holger, Bankiti, Varun, Lang, Alex H., Vora, Sourabh, Liong, 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)*.
- [93] Cai, Kaiwen, Wang, Bing, and Lu, Chris Xiaoxuan. 2022. AutoPlace: Robust Place Recognition with Single-chip Automotive Radar. Pages 2222–2228 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [94] Callmer, Jonas, Törnqvist, David, Gustafsson, Fredrik, Svensson, Henrik, and Carlbom, Pelle. 2011. Radar SLAM using visual features. *EURASIP Journal on Advances in Signal Processing*, **2011**(09), 1–11.
- [95] Calonder, Michael, Lepetit, Vincent, Strecha, Christoph, and Fua, Pascal. 2010. BRIEF: Binary Robust Independent Elementary Features. Pages 778–792 of: *European Conference on Computer Vision (ECCV)*.
- [96] 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.
- [97] Camurri, Marco, Fallon, Maurice, Bazeille, Stéphane, Radulescu, Andreea, Barasol, 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.
- [98] 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**.
- [99] 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.
- [100] Carlone, L. 2023. Estimation Contracts for Outlier-Robust Geometric Perception. *Foundations and Trends (FnT) in Robotics, arXiv preprint: 2208.10521*. .
- [101] Carlone, L., and Calafiore, G. 2018a. Convex Relaxations for Pose Graph Optimization with Outliers. *IEEE Robotics and Automation Letters*, **3**(2), 1160–1167. arxiv preprint: 1801.02112, .
- [102] Carlone, Luca, and Calafiore, Giuseppe C. 2018b. Convex Relaxations for Pose Graph Optimization with Outliers. *IEEE Robotics and Automation Letters*, **3**(2), 1160–1167.
- [103] 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.
- [104] 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.

- [105] Cazals, Frédéric, and Giesen, Joachim. 2006. *Delaunay Triangulation Based Surface Reconstruction*. Springer Berlin Heidelberg. Pages 231–276.
- [106] 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.
- [107] Censi, A. 2007. An accurate closed-form estimate of ICP’s covariance. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [108] Censi, Andrea, and Scaramuzza, Davide. 2014. Low-Latency Event-Based Visual Odometry. Pages 703–710 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [109] Chadwick, Jeffrey N., and Bindel, David S. 2015. An efficient solver for sparse linear systems based on rank-structured Cholesky factorization. *arXiv preprint arXiv:1507.05593*.
- [110] Chakravarthi, Bharatesh, Verma, Aayush Atul, Daniilidis, Kostas, Fermuller, Cornelia, and Yang, Yezhou. 2024. Recent Event Camera Innovations: A Survey. In: *European Conf. on Computer Vision Workshops*.
- [111] 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*.
- [112] 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 Carloni, 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. .
- [113] Chatfield, Averil B. 1997. *Fundamentals of High Accuracy Inertial Navigation*. AIAA.
- [114] Chatila, R., and Laumond, J.-P. 1985. Position referencing and consistent world modeling for mobile robots. Pages 138–145 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [115] Chatterjee, A., and Govindu, V. M. 2013. Efficient and Robust Large-Scale Rotation Averaging. Pages 521–528 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [116] Chebrolu, Nived, Läbe, Thomas, Vysotska, Olga, Behley, Jens, and Stachniss, Cyrill. 2020. Adaptive Robust Kernels for Non-Linear Least Squares Problems. *arXiv preprint arXiv:2004.14938*.
- [117] 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.
- [118] 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.
- [119] Chen, Chuchu, Geneva, Patrick, Peng, Yuxiang, Lee, Woosik, and Huang, Guoquan. 2023a. Optimization-Based VINS: Consistency, Marginalization, and FEJ. Pages 1517–1524 of: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

- [120] Chen, Chuchu, Peng, Yuxiang, and Huang, Guoquan. 2024a (May). Fast and Consistent Covariance Recovery for Sliding-window Optimization-based VINS. In: *Proc. International Conference on Robotics and Automation*.
- [121] 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)*.
- [122] Chen, Peiyu, Guan, Weipeng, and Lu, Peng. 2023b. ESVIO: Event-based Stereo Visual Inertial Odometry. *IEEE Robotics and Automation Letters*, 8(6), 3661–3668.
- [123] Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, and Zhang, Zheng. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.
- [124] Chen, Xiangyu, Yang, Fan, and Wang, Chen. 2024b. iA\*: Imperative Learning-based A\* Search for Pathfinding. *arXiv preprint arXiv:2403.15870*.
- [125] 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.
- [126] 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.
- [127] 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.
- [128] Cheng, Q., Zeller, N., and Cremers, D. 2022a. Vision-Based Large-scale 3D Semantic Mapping for Autonomous Driving Applications. Pages 9235–9242 of: *International Conference on Robotics and Automation (ICRA)*.
- [129] Cheng, Yuwei, Su, Jingran, Jiang, Mengxin, and Liu, Yimin. 2022b. A Novel Radar Point Cloud Generation Method for Robot Environment Perception. *IEEE Trans. Robotics*.
- [130] 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)*.
- [131] 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)*.
- [132] 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*.
- [133] Chirikjian, G. S. 2009. *Stochastic Models, Information Theory, and Lie Groups, Volume 1: Classical Results and Geometric Methods (Applied and Numerical Harmonic Analysis)*. Birkhauser.
- [134] Chirikjian, G. S. 2012. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications (Applied and Numerical Harmonic Analysis)*. Birkhauser.

- [135] Chirikjian, 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.
- [136] 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)*.
- [137] 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.
- [138] 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)*.
- [139] 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.
- [140] 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.
- [141] Chum, O., and Matas, J. 2005. Matching with PROSAC - Progressive Sample Consensus. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [142] Chum, Ondřej, Matas, Jiří, and Kittler, Josef. 2003. Locally optimized RANSAC. Pages 236–243 of: *Joint Pattern Recognition Symposium*. Springer.
- [143] 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).
- [144] 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.
- [145] Clark, J.H. 1976. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, **19**(1), 547–554.
- [146] Clark, Steve, and Durrant-Whyte, Hugh. 1998. Autonomous land vehicle navigation using millimeter wave radar. Pages 3697–3702 of: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 4. IEEE.
- [147] Cohen, Nadav, and Klein, Itzik. 2024. *Inertial Navigation Meets Deep Learning: A Survey of Current Trends and Future Directions*.
- [148] Collobert, Ronan, Bengio, Samy, and Mariéthoz, Johnny. 2002. *Torch: a modular machine learning software library*. Tech. rept. Idiap.
- [149] 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)*.
- [150] Crassidis, J.L. 2006. Sigma-point Kalman filtering for integrated GPS and inertial navigation. *IEEE Trans. Aerosp. Electron. Syst.*, **42**(2), 750–756.
- [151] 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)*.

- [152] 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)*.
- [153] 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.
- [154] 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.
- [155] 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.
- [156] 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)*.
- [157] Davison, Andrew J. 2018. FutureMapping: The Computational Structure of Spatial AI Systems. *arXiv preprint*, Mar.
- [158] De Martini, Daniele, Gadd, Matthew, and Newman, Paul. 2020. kRadar++: Coarse-to-Fine FMCW Scanning Radar Localisation. *Sensors*, **20**(21).
- [159] 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.
- [160] Dellaert, F. 2005. Square Root SAM: Simultaneous Location and Mapping via Square Root Information Smoothing. In: *Robotics: Science and Systems (RSS)*.
- [161] Dellaert, Frank. 2012. *Factor graphs and GTSAM: A hands-on introduction*. Tech. rept. Georgia Institute of Technology.
- [162] Dellaert, Frank, and Contributors, GTSAM. 2022 (May). *GTSAM: Georgia Tech Smoothing and Mapping Library*. Georgia Tech Borg Lab.
- [163] Dellaert, Frank, and Kaess, Michael. 2006. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, **25**(12), 1181–1203.
- [164] Dellaert, Frank, Kaess, Michael, et al. 2017. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, **6**(1-2), 1–139.
- [165] Dellenbach, P., Deschaud, J., Jacquet, B., and Goulette, F. 2022. CT-ICP Real-Time Elastic LiDAR Odometry with Loop Closure. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [166] 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)*.
- [167] 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)*.
- [168] 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)*.

- [169] Deng, Junyuan, Chen, Xieyuanli, Xia, Songpengcheng, Sun, Zhen, Liu, Guoqing, Yu, Wenxian, and Pei, Ling. 2023. NeRF-LOAM: Neural Implicit Representation for Large-Scale Incremental LiDAR Odometry and Mapping. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [170] Dennis, J.E., and Schnabel, R.B. 1983. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall.
- [171] Deschaud, J.-E. 2018. IMLS-SLAM: Scan-to-Model Matching Based on 3D Data. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [172] 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)*.
- [173] DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. 2018a. SuperPoint: Self-Supervised Interest Point Detection and Description. *arXiv:1712.07629 [cs]*, Apr.
- [174] DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. 2018b. SuperPoint: Self-supervised interest point detection and description. In: *IEEE Conference in Computer Vision and Pattern Recognition (CVPR) workshops*.
- [175] Diebel, James. 2006. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Tech. rept. Stanford University.
- [176] 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.
- [177] 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)*.
- [178] 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)*.
- [179] 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: Results of the 8th International Conference*. Springer.
- [180] Dosovitskiy, Alexey, Ros, German, Codevilla, Felipe, Lopez, Antonio, and Koltun, Vladlen. 2017. CARLA: An Open Urban Driving Simulator. In: *Conf. on Robot Learning (CoRL)*.
- [181] Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: *Intl. Conf. on Learning Representations (ICLR)*.
- [182] Droeschel, D., and Behnke, S. 2018a. Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [183] Droeschel, David, and Behnke, Sven. 2018b. Efficient continuous-time SLAM

- for 3D lidar-based online mapping. Pages 5000–5007 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [184] Duane, C Brown. 1971. Close-range camera calibration. *Photogramm. Eng.*, **37**(8), 855–866.
- [185] 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)*.
- [186] 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.
- [187] 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.
- [188] Duckett, T., Marsland, S., and Shapiro, J. 2002. Fast, On-line Learning of Globally Consistent Maps. *Autonomous Robots*, **12**(3), 287–300.
- [189] 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.
- [190] Dunkley, O., Engel, J., Sturm, J., and Cremers, D. 2014. Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor. In: *IROS2014 Aerial Open Source Robotics Workshop*.
- [191] Dunning, Iain, Huchette, Joey, and Lubin, Miles. 2017. JuMP: A modeling language for mathematical optimization. *SIAM review*, **59**(2), 295–320.
- [192] Durrant-Whyte, H.F. 1988. Uncertain geometry in robotics. *IEEE Trans. Robot. Automat.*, **4**(1), 23–31.
- [193] 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), *Robotics Research: The 7th International Symposium (ISRR 95)*. Springer-Verlag.
- [194] Dusmanu, Mihai, Rocco, Ignacio, Pajdla, Tomas, Sivic, Josef, Pollefeys, Marc, Mikulik, Andrej, and Sattler, Torsten. 2019. D2-net: A trainable CNN for joint description and detection of local features. In: *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*.
- [195] 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)*.
- [196] Ebadi, Kamak, Bernreiter, Lukas, Biggie, Harel, Catt, Gavin, Chang, Yun, Chatterjee, Arghya, Denniston, Christopher E., Deschênes, Simon-Pierre, Harlow, Kyle, Khattak, Shehryar, Nogueira, Lucas, Palieri, Matteo, Petráček, Pavel, Petrlík, Matěj, Reinke, Andrzej, Krátký, Vít, Zhao, Shibo, Aghamohammadi, Ali-akbar, Alexis, Kostas, Heckman, Christoffer, Khosoussi, Kasra, Kottege, Navinda, Morrell, Benjamin, Hutter, Marco, Pauling, Fred, Pomerleau, François, Saska, Martin, Scherer, Sebastian, Siegwart, Roland, Williams, Jason L., and Carbone, Luca. 2024a. Present and Future of SLAM in Extreme Environments: The DARPA SubT Challenge. *IEEE Trans. Robotics*, **40**, 936–959.

- [197] Ebadi, Kamak, Bernreiter, Lukas, Biggie, Harel, Catt, Gavin, Chang, Yun, Chatterjee, Arghya, Denniston, Christopher E., Deschênes, Simon-Pierre, Harlow, Kyle, Khattak, Shehryar, Nogueira, Lucas, Palieri, Matteo, Petracek, Pavel, Petrlík, Matej, Reinke, Andrzej, Kratky, Vít, Zhao, Shibo, akbar Aghamohammadi, Ali, Alexis, Kostas, Heckman, Christoffer, Khosoussi, Kasra, Kottege, Navinda, Morrell, Benjamin, Hutter, Marco, Pauling, Fred, Pomerleau, François, Saska, Martin, Scherer, Sebastian, Siegwart, Roland, Williams, Jason L., and Carbone, Luca. 2024b. Present and Future of SLAM in Extreme Underground Environments. *IEEE Trans. Robotics*, **40**, 936–959.
- [198] 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.
- [199] Eckenhoff, Kevin, Geneva, Patrick, and Huang, Guoquan. 2021. MIMC-VINS: A Versatile and Resilient Multi-IMU Multi-Camera Visual-Inertial Navigation System. *IEEE Transactions on Robotics*, **37**(5), 1360–1380.
- [200] 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)*.
- [201] 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)*.
- [202] Elfes, A. 1989. Using occupancy grids for mobile robot perception and navigation. *Computer*, **22**(6), 46–57.
- [203] Elseberg, J., Borrman, 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.
- [204] Engel, J., Sturm, J., and Cremers, D. 2012 (Oct.). Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing. In: *Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMor) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*.
- [205] Engel, Jakob, Schöps, Thomas, and Cremers, Daniel. 2014. LSD-SLAM: Large-scale direct monocular SLAM. Pages 834–849 of: *Computer Vision-ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part II 13*. Springer.
- [206] Engel, Jakob, Koltun, Vladlen, and Cremers, Daniel. 2018a. Direct sparse odometry. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(3), 611–625.
- [207] Engel, Jakob, Koltun, Vladlen, and Cremers, Daniel. 2018b. Direct Sparse Odometry. *IEEE Trans. Pattern Anal. Machine Intell.*, **40**(3), 611–625.
- [208] Enqvist, O., Josephson, K., and Kahl, F. 2009. Optimal correspondences from pairwise constraints. Pages 1295–1302 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [209] Eustice, R., Singh, H., and Leonard, J. 2005a (April). Exactly Sparse Delayed-State Filters. Pages 2417–2424 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [210] Eustice, R., Walter, M., and Leonard, J. 2005b (Aug). Sparse Extended Information Filters: Insights into Sparsification. Pages 3281–3288 of: *IEEE/RJS Intl. Conf. on Intelligent Robots and Systems (IROS)*.

- [211] Fahmi, Shamel, Fink, Geoff, and Semini, Claudio. 2021. On State Estimation for Legged Locomotion Over Soft Terrain. *IEEE Sensors Letters*, **5**(1), 1–4.
- [212] 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*.
- [213] 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*.
- [214] 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.
- [215] Farrell, J.A. 2008. *Aided Navigation: GPS with High Rate Sensors*. McGraw-Hill.
- [216] Featherstone, Roy. 2007. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag.
- [217] 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*.
- [218] Fent, Felix, Kuttenreich, 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)*.
- [219] 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  $1280 \times 720$  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)*.
- [220] Fischler, M., and Bolles, R. 1981a. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Commun. ACM*, **24**, 381–395.
- [221] Fischler, Martin A., and Bolles, Robert C. 1981b. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24**(6), 381–395.
- [222] 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.
- [223] 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.
- [224] 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*.
- [225] Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. 2017. On-Manifold Preintegration for Real-Time Visual-Inertial Odometry. *IEEE Transactions on Robotics*, **33**(1), 1–21.
- [226] Forster, Christian, Carlone, 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)*.
- [227] Forster, Christian, Carlone, Luca, Dellaert, Frank, and Scaramuzza, Davide. 2016. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, **33**(1), 1–21.
- [228] Fourie, Dehann. 2017. *Multi-modal and inertial sensor solutions for navigation-type factor graphs*. Ph.D. thesis, MIT.
- [229] Frese, U. 2005. Treemap: An  $O(\log n)$  Algorithm for Simultaneous Localization and Mapping. Pages 455–476 of: *Spatial Cognition IV*. Springer Verlag.
- [230] Frese, U., Larsson, P., and Duckett, T. 2005. A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping. *IEEE Trans. Robotics*, **21**(2), 196–207.
- [231] Fritzsche, Paul, Kueppers, Simon, Briese, Gunnar, and Wagner, Bernardo. 2016. Radar and LiDAR Sensorfusion in Low Visibility Environments. Pages 30–36 of: *ICINCO*.
- [232] Fritzsche, Paul, Kueppers, Simon, Briese, Gunnar, and Wagner, Bernardo. 2018. *Fusing LiDAR and Radar Data to Perform SLAM in Harsh Environments*. Cham: Springer International Publishing. Pages 175–189.
- [233] Fu, Taimeng, Su, Shaoshu, Lu, Yiren, and Wang, Chen. 2024. iSLAM: Imperative SLAM. *IEEE Robotics and Automation Letters (RA-L)*.
- [234] Funk, Nils, Tarrio, Juan, Papatheodorou, Sotiris, Popović, Marija, Alcantarilla, Pablo F., and Leutenegger, Stefan. 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.
- [235] 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.
- [236] Furgale, Paul, Rehder, Joern, and Siegwart, Roland. 2013. Unified temporal and spatial calibration for multi-sensor systems. Pages 1280–1286 of: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.
- [237] 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)*.
- [238] Gadd, Matthew, De Martini, Daniele, Bartlett, Oliver, Murcatt, Paul, Towleson, Matt, Widojo, Matthew, Mušat, Valentina, Robinson, Luke, Panagiotaki, Efimia, Pramatarov, Georgi, et al. 2024. OORD: The Oxford Offroad Radar Dataset. *arXiv preprint arXiv:2403.02845*.
- [239] 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)*.

- [240] Gallego, Guillermo, and Scaramuzza, Davide. 2017. Accurate Angular Velocity Estimation with an Event Camera. *IEEE Robotics and Automation Letters*, **2**(2), 632–639.
- [241] 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*.
- [242] 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.
- [243] 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)*.
- [244] 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)*.
- [245] 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.
- [246] Gálvez-López, Dorian, and Tardós, J. D. 2012. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Trans. Robotics*, **28**(5), 1188–1197.
- [247] Gálvez-López, Dorian, and Tardos, Juan D. 2012. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on robotics*, **28**(5), 1188–1197.
- [248] 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.
- [249] 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)*.
- [250] 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)*.
- [251] 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)*.
- [252] 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.
- [253] 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)*.

- [254] Geneva, Patrick, Eckenhoff, Kevin, Yang, Yulin, and Huang, Guoquan. 2018a. LIPS: LiDAR-Inertial 3D Plane SLAM. Pages 123–130 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [255] Geneva, Patrick, Eckenhoff, Kevin, Yang, Yulin, and Huang, Guoquan. 2018b (Oct.). LIPS: LiDAR-Inertial 3D Plane SLAM. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- [256] Geneva, Patrick, Eckenhoff, Kevin, Lee, Woosik, Yang, Yulin, and Huang, Guoquan. 2020. OpenVINS: A Research Platform for Visual-Inertial Estimation. In: *Proc. of the IEEE International Conference on Robotics and Automation*.
- [257] 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.
- [258] 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)*.
- [259] 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.
- [260] Ghosh, Suman, and Gallego, Guillermo. 2024. Event-based Stereo Depth Estimation: A Survey. *arXiv preprint*.
- [261] Ghosh, Suman, Cavinato, Valentina, and Gallego, Guillermo. 2024. ES-PTAM: Event-based Stereo Parallel Tracking and Mapping. In: *European Conf. on Computer Vision Workshops*.
- [262] 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.
- [263] Giseop Kim, Ayoung Kim. 2022. LT-mapper: A Modular Framework for LiDAR-based Lifelong Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [264] Gladkova, M, Wang, R, Zeller, N, and Cremers, D. 2021. Tight Integration of Feature-based Relocalization in Monocular Direct Visual Odometry. In: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.
- [265] Golub, G.H., and Loan, C.F. Van. 1996. *Matrix Computations*. Third edn. Baltimore: Johns Hopkins University Press.
- [266] Gomez, Jorge, Patel, Saavan, Sarwar, Syed Shakib, Li, Ziyun, Capoccia, Raf-faele, 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*.
- [267] 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.
- [268] Grinvald, Margarita, Furrer, Fadri, Novkovic, Tonci, Chung, Jen Jen, Cadena, 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.
- [269] 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.
- [270] 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)*.
- [271] 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)*.
- [272] Grupp, Michael. 2017. *evo: Python package for the evaluation of odometry and SLAM*. <https://github.com/MichaelGrupp/evo>.
- [273] Guan, Weipeng, Lin, Fuling, Chen, Peiyu, and Lu, Peng. 2024a. DEIO: Deep Event Inertial Odometry. *arXiv preprint*.
- [274] 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.
- [275] Guan, Weipeng, Chen, Peiyu, Xie, Yuhan, 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.
- [276] Guennebaud, Gaël, Jacob, Benoit, et al. 2010. Eigen. URL: <http://eigen.tuxfamily.org>, **3**.
- [277] 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)*.
- [278] 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.
- [279] 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.
- [280] Guo, Shuang, and Gallego, Guillermo. 2024b. Event-based Mosaicing Bundle Adjustment. Pages 479–496 of: *European Conf. on Computer Vision (ECCV)*.
- [281] Guo, Shuang, and Gallego, Guillermo. 2024c. Event-based Photometric Bundle Adjustment. *arXiv preprint*.
- [282] 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)*.
- [283] 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)*.

- [284] Hadžiger, 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 Robotics (ECMR)*.
- [285] 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)*.
- [286] Handa, Ankur, Bloesch, Michael, Pătrăucean, Viorica, Stent, Simon, MacCormac, John, and Davison, Andrew. 2016. gvnn: Neural network library for geometric computer vision. Pages 67–82 of: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8–10 and 15–16, 2016, Proceedings, Part III* 14. Springer.
- [287] Harris, C., and Stephens, M. 1988. A combined corner and edge detector. Pages 147–151 of: *Proceedings of the Alvey Vision Conference*.
- [288] 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)*.
- [289] Hart, William E, Laird, Carl D, Watson, Jean-Paul, Woodruff, David L, Hackebeil, Gabriel A, Nicholson, Bethany L, Sirola, John D, et al. 2017. *Pyomo-optimization modeling in python*. Vol. 67. Springer.
- [290] Hartley, R., and Zisserman, A. 2000. *Multiple View Geometry in Computer Vision*. Cambridge University Press.
- [291] Hartley, R.I., and Kahl, F. 2009. Global optimization through rotation space search. *Intl. J. of Computer Vision*, **82**(1), 64–79.
- [292] Hartley, Richard, and Zisserman, Andrew. 2003. *Multiple View Geometry in Computer Vision*. Cambridge University Press. 2nd Edition.
- [293] 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)*.
- [294] 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)*.
- [295] 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)*.
- [296] Hartley, Ross, Ghaffari, Maani, Eustice, Ryan M, and Grizzle, Jessy W. 2020a. Contact-aided invariant extended Kalman filtering for robot state estimation. *Intl. J. of Robotics Research*, **39**(4), 402–430.
- [297] Hartley, Ross, Ghaffari, Maani, Eustice, Ryan M, and Grizzle, Jessy W. 2020b. Contact-aided invariant extended Kalman filtering for robot state estimation. *Intl. J. of Robotics Research*, **39**(4), 402–430.
- [298] 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.
- [299] He, Li, Wang, Xiaolong, and Zhang, Hong. 2016. 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.
- [300] 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.
- [301] Herath, Sachini, Yan, Hang, and Furukawa, Yasutaka. 2020. Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods. Pages 3146–3152 of: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE.
- [302] Herath, Sachini, Caruso, David, Liu, Chen, Chen, Yufan, and Furukawa, Yasutaka. 2022. Neural inertial localization. Pages 6604–6613 of: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- [303] Hermann, R., and Krener, A. 1977. Nonlinear controllability and observability. *IEEE Trans. on Automatic Control*, **22**(5), 728–740.
- [304] 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*.
- [305] Hesch, J. A., Mirzaei, F. M., Mariottini, G. L., and Roumeliotis, S. I. 2010 (May. 3 - 8). A Laser-Aided Inertial Navigation System (L-INS) for human localization in unknown indoor environments. Pages 5376–5382 of: *International Conference on Robotics and Automation*.
- [306] 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.
- [307] 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.
- [308] 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.
- [309] 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.
- [310] 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)*.
- [311] 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)*.
- [312] 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.
- [313] Hoeller, David, Rudin, Nikita, Sako, Dhionis V., and Hutter, Marco. 2024. ANYmal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, **9**(88).

- [314] Holmstrom, Sven T. S., Baran, Utku, and Urey, Hakan. 2014. MEMS Laser Scanners: A Review. *Journal of Microelectromechanical Systems*, **23**(2), 259–275.
- [315] 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: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I* 14. Springer.
- [316] 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.
- [317] 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)*.
- [318] 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.
- [319] Hong, Ziyang, Petillot, Yvan, Zhang, Kaicheng, Xu, Shida, and Wang, Sen. 2023. Large-Scale Radar Localization using Online Public Maps. Pages 3990–3996 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [320] 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.
- [321] Horn, Berthold K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, **4**(4), 629–642.
- [322] 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.
- [323] 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*.
- [324] 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 arXiv:2407.11705*.
- [325] Huang, Guoquan. 2017. Towards Consistent Filtering for Discrete-Time Partially-Observable Nonlinear Systems. *Systems and Control Letters*, **106**(Aug.), 87–95.
- [326] Huang, Guoquan, Mourikis, Anastasios I., and Roumeliotis, Stergios I. 2010. Observability-based Rules for Designing Consistent EKF SLAM Estimators. *International Journal of Robotics Research*, **29**(5), 502–528.
- [327] Huang, Guoquan, Mourikis, Anastasios I., and Roumeliotis, Stergios I. 2011 (Sept.). An Observability Constrained Sliding Window Filter for SLAM. Pages 65–72 of: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- [328] Huang, J.-T., Xu, R., Hinduja, A., and Kaess, M. 2024 (May). Multi-Radar

- Inertial Odometry for 3D State Estimation using mmWave Imaging Radar. In: *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [329] Huang, Yewei, Shan, Tixiao, Chen, Fanfei, and Englot, Brendan. 2022. 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.
- [330] Huber, P. 1981. *Robust Statistics*. John Wiley & Sons, New York, NY.
- [331] 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)*.
- [332] 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).
- [333] 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. *International Journal of Remote Sensing*, **29**(5), 1339–1366.
- [334] ICRA Quadruped Robot Challenge. 2024. *ICRA Quadruped Robot Challenge*. <https://quadruped-robot-challenges.notion.site/Quadruped-Robot-Challenges-bdc4af35638c4036817c3212e602b0e3>. [Online; accessed 10-Jun-2024].
- [335] Indelman, V., Williams, S., Kaess, M., and Dellaert, F. 2012. Factor Graph Based Incremental Smoothing in Inertial Navigation Systems. In: *Intl. Conf. on Information Fusion (FUSION)*.
- [336] 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.
- [337] 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*.
- [338] 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)*.
- [339] Izquierdo, Sergio, and Civera, Javier. 2024. Optimal transport aggregation for visual place recognition. Pages 17658–17668 of: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- [340] 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)*.
- [341] 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.
- [342] 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.

- [343] 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. *The International Journal of Robotics Research*, **38**(6), 642–657.
- [344] 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.
- [345] Ji, Kaiyi, Yang, Junjie, and Liang, Yingbin. 2021. Bilevel optimization: Convergence analysis and enhanced design. Pages 4882–4892 of: *International conference on machine learning*. PMLR.
- [346] 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*.
- [347] Jiang, Huaiyu, 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)*.
- [348] 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)*.
- [349] 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.
- [350] 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*. (arXiv:2402.00399 [cs.RO]).
- [351] 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.
- [352] Kaess, M., Ranganathan, A., and Dellaert, F. 2008. iSAM: Incremental Smoothing and Mapping. *IEEE Trans. Robotics*, **24**(6), 1365–1378.
- [353] 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)*.
- [354] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., and Dellaert, F. 2012. iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *Intl. J. of Robotics Research, IJRR*, **31**(2), 216–235.
- [355] Kaess, Michael. 2015. Simultaneous localization and mapping with infinite planes. Pages 4605–4611 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

- [356] 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)*.
- [357] 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.
- [358] Kannala, Juho, and Brandt, Sami S. 2006. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, **28**(8), 1335–1340.
- [359] 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)*.
- [360] 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.
- [361] 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.
- [362] Keetha, Nikhil, Karhade, Jay, Jatavallabhula, Krishna Murthy, Yang, Gengshan, Scherer, Sebastian, Ramanan, Deva, and Luiten, Jonathon. 2023. SplatTAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM. *arXiv preprint*.
- [363] Keller, M., Lefloch, D., Lambers, M., and Izadi, S. 2013. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In: *Intl. Conf. on 3D Vision (3DV)*.
- [364] 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)*.
- [365] 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)*.
- [366] Kerbl, Bernhard, Kopanas, Georgios, Leimkühler, Thomas, and Drettakis, George. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *arXiv preprint*.
- [367] Kerl, C., Sturm, J., and Cremers, D. 2013a. Dense Visual SLAM for RGB-D Cameras. In: *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*.
- [368] Kerl, C., Sturm, J., and Cremers, D. 2013b (May). Robust Odometry Estimation for RGB-D Cameras. In: *International Conference on Robotics and Automation (ICRA)*.
- [369] Kerl, Christian, Sturm, Jürgen, and Cremers, Daniel. 2013c. Robust odometry estimation for RGB-D cameras. Pages 3748–3754 of: *2013 IEEE international conference on robotics and automation*. IEEE.
- [370] Khader, Motaz, and Cherian, Samir. 2020. An introduction to automotive lidar. *Texas Instruments*.

- [371] 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)*.
- [372] 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)*.
- [373] Kim, Giseop, and Kim, Ayoung. 2018a. 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.
- [374] Kim, Giseop, and Kim, Ayoung. 2018b (10). Scan Context: Egocentric Spatial Descriptor for Place Recognition within 3D Point Cloud Map. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [375] Kim, Giseop, Park, Byungjae, and Kim, Ayoung. 2019. 1-day learning, 1-year localization: Long-term lidar localization using scan context image. *IEEE Robotics and Automation Letters*, **4**(2), 1948–1955.
- [376] Kim, Giseop, Park, Yeong Sang, Cho, Younghun, Jeong, Jinyong, and Kim, Ayoung. 2020a. MuLRan: Multimodal Range Dataset for Urban Place Recognition. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. Paris: IEEE.
- [377] Kim, Giseop, Park, Yeong Sang, Cho, Younghun, Jeong, Jinyong, and Kim, Ayoung. 2020b. Mulran: Multimodal range dataset for urban place recognition. Pages 6246–6253 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [378] Kim, Giseop, Choi, Sunwook, and Kim, Ayoung. 2022a. Scan context++: Structural place recognition robust to rotation and lateral variations in urban environments. *IEEE Trans. Robotics*, **38**(3), 1856–1874.
- [379] Kim, Giseop, Choi, Sunwook, and Kim, Ayoung. 2022b. Scan Context++: Structural Place Recognition Robust to Rotation and Lateral Variations in Urban Environments. *IEEE Trans. Robotics*, **38**(3), 1856–1874.
- [380] 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)*.
- [381] 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)*.
- [382] 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)*.
- [383] 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.
- [384] 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)*.
- [385] 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*.
- [386] Kim, Soohwan, and Kim, Jonghyuk. 2015. *GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes*. Springer International Publishing. Pages 319–332.
- [387] Kim, Yeeun, Yu, Byeongho, Lee, Eungchang Mason, Kim, Joon-ha, Park, Hae-won, and Myung, Hyun. 2022c. STEP: State Estimator for Legged Robots Using a Preintegrated Foot Velocity Factor. *IEEE Robotics and Automation Letters*, **7**(2), 4456–4463.
- [388] Kingma, Diederik P, and Ba, Jimmy. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [389] Klein, G., and Murray, D. 2007. Parallel tracking and mapping for small AR workspaces. Pages 225–234 of: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*.
- [390] 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)*.
- [391] Klenk, Simon, Motzett, Marvin, Koestler, Lukas, and Cremers, Daniel. 2024. Deep Event Visual Odometry. Pages 739–749 of: *Intl. Conf. on 3D Vision (3DV)*.
- [392] Koenemann, Jonas, Licitra, Giovanni, Alp, Mustafa, and Diehl, Moritz. 2017. *Openocl—open optimal control library*.
- [393] 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.
- [394] Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.
- [395] Konolige, K. 2004. Large-scale map-making. In: *Proc. 21<sup>th</sup> AAAI National Conference on AI*.
- [396] 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.
- [397] Koskinen, Markku, Kostamovaara, Juha Tapio, 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.
- [398] Kottas, Dimitrios, and Roumeliotis, Stergios. 2013 (June 24–28.). Exploiting Urban Scenes for Vision-aided Inertial Navigation. In: *Robotics: Science and Systems (RSS)*.
- [399] 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.
- [400] 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.
- [401] Kramer, Andrew, and Heckman, Christoffer. 2020. Radar-Inertial State Estimation and Obstacle Detection for Micro-Aerial Vehicles in Dense Fog. Pages 3–16 of: *International Symposium on Experimental Robotics*. Springer.
- [402] Kramer, Andrew, Stahoviak, Carl, Santamaria-Navarro, Angel, Aghamohammadi, 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.
- [403] 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.
- [404] Krizhevsky, A., Sutskever, I., and Hinton, G. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: *Conf. Neural Information Processing Systems (NIPS)*.
- [405] 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)*.
- [406] 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)*.
- [407] Kukko, Antero, Kaijaluoto, Risto, Kaartinen, Harri, Lehtola, Ville V, Jaakkola, Anttoni, and Hyypä, Juha. 2017. Graph SLAM correction for single scanner MLS forest data under boreal forest canopy. *ISPRS Journal of Photogrammetry and Remote Sensing*, **132**, 199–209.
- [408] Kümmerle, 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)*.
- [409] 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)*.
- [410] 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*. extended ArXiv version: , Supplemental Material: .
- [411] 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)*.
- [412] Latif, Y., Lerma, C. D. C., and Neira, J. 2012. Robust Loop Closing Over Time. In: *Robotics: Science and Systems (RSS)*.
- [413] 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.
- [414] Le Gentil, C., and Vidal-Calleja, T. 2021. Continuous Integration over SO(3) for IMU Preintegration. In: *Robotics: Science and Systems (RSS)*.
- [415] 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.
- [416] 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.
- [417] Le Gentil, Cedric, Vidal-Calleja, Teresa, and Huang, Shoudong. 2021. IN2LAAMA: Inertial Lidar Localization Autocalibration and Mapping. *IEEE Transactions on Robotics*, **37**(1), 275–290.
- [418] 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.
- [419] 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.
- [420] 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)*.
- [421] 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)*.
- [422] Lee, Joonho, Hwangbo, Jemin, Wellhausen, Lorenz, Koltun, Vladlen, and Hutter, Marco. 2020. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, **5**(47), 5986.
- [423] 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.
- [424] Lepora, Nathan F., and Lloyd, John. 2020. Optimal Deep Learning for Robot Touch: Training Accurate Pose Models of 3D Surfaces and Edges. *IEEE Robotics & Automation Magazine*, **27**(2), 66–77.
- [425] Leroy, Vincent, Cabon, Yohann, and Revaud, Jerome. 2024. Grounding Image Matching in 3D with MAST3R. Page 71–91 of: *European Conference on Computer Vision (ECCV)*.
- [426] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. 2015a. Keyframe-based visual-inertial odometry using nonlinear optimization. *Intl. J. of Robotics Research*, **34**(3), 314–334.
- [427] Leutenegger, Stefan. 2022. Okvis2: Realtime scalable visual-inertial slam with loop closure. *arXiv preprint arXiv:2202.09199*.
- [428] Leutenegger, Stefan, Chli, Margarita, and Siegwart, Roland Y. 2011. BRISK: Binary Robust Invariant Scalable Keypoints. Pages 2548–2555 of: *International Conference on Computer Vision (ICCV)*.
- [429] Leutenegger, Stefan, Lynen, Simon, Bosse, Michael, Siegwart, Roland, and Furgale, Paul. 2015b. Keyframe-based visual-inertial odometry using nonlinear optimization. *Intl. J. of Robotics Research*, **34**(3), 314–334.
- [430] Levenberg, K. 1944. A Method for the Solution of Certain Nonlinear Problems in Least Squares. *Quart. Appl. Math.*, **2**(2), 164–168.
- [431] 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)*.
- [432] Li, Fangting, Zhang, Guoqiang, and Yan, Jun. 2008. Coregistration Based on Sift Algorithm for Synthetic Aperture Radar Interferometry.
- [433] Li, H. 2009. Consensus set maximization with guaranteed global optimality for robust geometry estimation. Pages 1074–1080 of: *Intl. Conf. on Computer Vision (ICCV)*.
- [434] Li, M., and Mourikis, A. 2013. High-Precision, Consistent EKF-based Visual-Inertial Odometry. *Intl. J. of Robotics Research*, **32**(6), 690–711.
- [435] 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)*.
- [436] 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)*.
- [437] 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.
- [438] 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.
- [439] Li, Xudong, Wang, Zhixiang, Liu, Zihao, Zhang, Yizhai, Zhang, Fan, Yao, Xiuming, and Huang, Panfeng. 2024. 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)*.
- [440] Lichtsteiner, Patrick, Posch, Christoph, and Delbrück, 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.
- [441] Lim, Hyungtae, Kim, Beomsoo, Kim, Daebeom, Lee, Eungchang Mason, and Myung, Hyun. 2024. Quattro++: Robust global registration exploiting ground segmentation for loop closing in LiDAR SLAM. *Intl. J. of Robotics Research*, **43**(5), 685–715.
- [442] 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)*.
- [443] 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)*.
- [444] 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.
- [445] 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.
- [446] Liston, Ronald A. 1967. Walking machine studies. *The Military Engineer*, **59**(388), 101–104.

- [447] Liu, Hanxiao, Simonyan, Karen, and Yang, Yiming. 2019. Darts: Differentiable architecture search. In: *International Conference on Learning Representations (ICLR)*.
- [448] 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.
- [449] 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 Transactions on Pattern Analysis and Machine Intelligence*, **44**(12), 10045–10067.
- [450] 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.
- [451] Liu, Xiaoye. 2008. Airborne LiDAR for DEM generation: some critical issues. *Progress in Physical Geography: Earth and Environment*, **32**(1), 31–49.
- [452] 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.
- [453] Liu, Zhe, Shi, Dianxi, Li, Ruihao, and Yang, Shaowu. 2023b. ESVIO: Event-Based Stereo Visual-Inertial Odometry. *Sensors*, **23**(4).
- [454] Lobo, Jorge, and Dias, Jorge. 2007. Relative Pose Calibration Between Visual and Inertial Sensors. *Intl. J. of Robotics Research*, **26**(6), 561–575.
- [455] 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: *Proceedings of the IEEE/CVF International Conference on Computer Vision*.
- [456] 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: *Proceedings of Robotics: Science and Systems*.
- [457] Loop, C., Cai, Q., Orts-Escalano, 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)*.
- [458] 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.
- [459] Lourakis, Manolis LA, and Argyros, Antonis A. 2005a. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? Pages 1526–1531 of: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2. IEEE.
- [460] Lourakis, MLA, and Argyros, Antonis A. 2005b. 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.
- [461] Lowe, D.G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Intl. J. of Computer Vision*, **60**(2), 91–110.
- [462] Lu, F., and Milios, E. 1997a. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, Apr, 333–349.

- [463] Lu, F., and Milios, E. 1997b. Robot pose estimation in unknown environments by matching 2D range scans. *J. of Intelligent and Robotic Systems*, April, 249:275.
- [464] Lu, Feng, and Milios, Evangelos. 1997c. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, **4**, 333–349.
- [465] 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.
- [466] 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.
- [467] 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.
- [468] 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)*.
- [469] Lusk, Parker C., Roy, Ronak, Fathian, Kaveh, and How, Jonathan P. 2021b (Nov.). MIXER: A Principled Framework for Multimodal, Multiway Data Association.
- [470] 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.
- [471] 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)*.
- [472] 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.
- [473] 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)*.
- [474] Lynch, Kevin M., and Park, Frank C. 2017a. *Modern Robotics - Mechanics, Planning, and Control*. USA: Cambridge University Press. Chap. 4, pages 137–152.
- [475] Lynch, Kevin M., and Park, Frank C. 2017b. *Modern Robotics - Mechanics, Planning, and Control*. USA: Cambridge University Press. Chap. 5, pages 171–190.
- [476] Lynch, Kevin M, Marchuk, Nicholas D, and Elwin, Matthew L. 2015. *Embedded Computing and Mechatronics with the PIC32 Microcontroller*. 1 edn. Newness. Chap. 21.
- [477] 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.

- [478] 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).
- [479] 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.
- [480] 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.
- [481] 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)*.
- [482] 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)*.
- [483] 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*.
- [484] Marquardt, D.W. 1963. An Algorithm for Least-Squares Estimation of Non-linear Parameters. *J. Soc. Indust. Appl. Math.*, **11**(2), 431–441.
- [485] 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.
- [486] 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)*.
- [487] MATLAB. 2010. *version 7.10.0 (R2010a)*. Natick, Massachusetts: The Math-Works Inc.
- [488] Matsuki, Hidenobu, von Stumberg, Lukas, Usenko, Vladyslav, Stückler, Jörg, and Cremers, Daniel. 2018. Omnidirectional DSO: Direct Sparse Odometry with Fisheye Cameras. *IEEE Robotics and Automation Letters*, **3**(4), 3693–3700.
- [489] Matsuki, Hidenobu, Murai2, Riku, Kelly, Paul H. J., and Davison, Andrew J. 2023. Gaussian Splatting SLAM. *arXiv preprint*.
- [490] Maybeck, P. 1979. *Stochastic Models, Estimation and Control*. Vol. 1. New York: Academic Press.
- [491] 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. European Conference on Mobile Robots (ECMR 2011).
- [492] McGhee, Robert B. 1968. Some finite state aspects of legged locomotion. *Mathematical Biosciences*, **2**(1-2), 67–84.
- [493] 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.

- [494] 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).
- [495] Medioni, G., Lee, M.-S., , and Tang, C.-K. 2000. *A Computational Framework for Segmentation and Grouping*. Elsevier.
- [496] 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)*.
- [497] Messikommer, Nico, Fang, Carter, Gehrig, Mathias, and Scaramuzza, Davide. 2023. Data-driven Feature Tracking for Event Cameras. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [498] 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 Robotics (ECMR)*.
- [499] 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.
- [500] Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., and Ng, R. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In: *European Conf. on Computer Vision (ECCV)*.
- [501] 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.
- [502] 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)*.
- [503] Moeys, 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.
- [504] 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*.
- [505] 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., Pfleuger, 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.
- [506] 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)*.
- [507] Mosher, Ralph S. 1969. Exploring the potential of a quadruped. *SAE Transactions*, 836–843.

- [508] Mourikis, A.I., and Roumeliotis, S.I. 2007 (April). A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. Pages 3565–3572 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [509] 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.
- [510] 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.
- [511] Mullane, J., Vo, B-N., Adams, M., and Vo, B-T. 2011. A Random-Finite-Set Approach to Bayesian SLAM. *IEEE Trans. Robotics*, **27**(2), 268–282.
- [512] 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)*. IEEE.
- [513] 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.
- [514] Mur-Artal, R., and Tardós, J. D. 2017. Visual-Inertial Monocular SLAM With Map Reuse. *IEEE Robotics and Automation Letters*, **2**(2), 796–803.
- [515] Mur-Artal, Raúl, and Tardós, Juan D. 2017a. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robotics*, **33**(5), 1255–1262.
- [516] Mur-Artal, Raúl, and Tardós, Juan D. 2017b. Visual-inertial monocular SLAM with map reuse. *IEEE Robotics and Automation Letters*, **2**(2), 796–803.
- [517] Mur-Artal, Raul, Montiel, Jose Maria Martinez, and Tardos, Juan D. 2015a. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. Robotics*, **31**(5), 1147–1163.
- [518] Mur-Artal, Raúl, Montiel, Jose M. M., and Tardós, Juan D. 2015b. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. Robotics*, **31**(5), 1147–1163.
- [519] Mur-Artal, Raúl, Montiel, José M. M., and Tardós, Juan D. 2015c. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Trans. Robotics*, **31**(5), 1147–1163.
- [520] Murai, Riku, Dexheimer, Eric, and Davison, Andrew J. 2024. MASt3R-SLAM: Real-Time Dense SLAM with 3D Reconstruction Priors. *arXiv preprint*.
- [521] Murray, R.M., Li, Z., and Sastry, S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.
- [522] 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.
- [523] Museth, Ken, Lait, Jeff, Johanson, John, Budsberg, Jeff, Henderson, Ron, Alden, Mihai, Cucka, Peter, Hill, David, and Pearce, Andrew. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. ACM.

- [524] Nabati, Ramin, and Qi, Hairong. 2020. Radar-camera sensor fusion for joint object detection and distance estimation in autonomous vehicles. *arXiv preprint arXiv:2009.08428*.
- [525] 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)*.
- [526] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. 2011a. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*.
- [527] Newcombe, Richard A, Lovegrove, Steven J, and Davison, Andrew J. 2011b. DTAM: Dense tracking and mapping in real-time. Pages 2320–2327 of: *2011 international conference on computer vision*. IEEE.
- [528] Newcombe, Richard A, Izadi, Shahram, Hilliges, Otmar, Molyneaux, David, Kim, David, Davison, Andrew J, Kohli, Pushmeet, Shotton, Jamie, Hodges, Steve, and Fitzgibbon, Andrew. 2011c. Kinectfusion: Real-time dense surface mapping and tracking. Pages 127–136 of: *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee.
- [529] 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)*.
- [530] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. 2013a. Real-time 3D Reconstruction at Scale using Voxel Hashing. In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH) Asia*.
- [531] Nießner, Matthias, Zollhöfer, Michael, Izadi, Shahram, and Stamminger, Marc. 2013b. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, **32**(6), 1–11.
- [532] 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)*.
- [533] Nistér, D. 2003. An Efficient Solution to the Five-Point Relative Pose Problem. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [534] Nitzberg, Ramon. 1972. Constant-false-alarm-rate signal processors for several types of interference. *IEEE Trans. Aerosp. Electron. Syst.*, 27–34.
- [535] 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*.
- [536] 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.
- [537] Nocedal, Jorge, and Wright, Stephen J. 1999. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag.
- [538] Nuss, Dominik, Reuter, Stephan, Thom, Markus, Yuan, Ting, Krehl, Gunther, 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.
- [539] O’Callaghan, Simon T, and Ramos, Fabio T. 2012. Gaussian process occupancy maps. *Intl. J. of Robotics Research*, **31**(1), 42–62.

- [540] 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.
- [541] 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.
- [542] Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., and Nieto, J. 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)*.
- [543] Oliphant, Travis E. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.
- [544] 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)*.
- [545] Olson, Edwin. 2009. Real-Time Correlative Scan Matching. Pages 4387–4393 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [546] Olson, Edwin. 2015. M3RSM: Many-to-many multi-resolution scan matching. Pages 5815–5821 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [547] Olson, Edwin, and Agarwal, Pratik. 2012 (July). Inference on networks of mixtures for robust robot mapping. In: *Robotics: Science and Systems (RSS)*.
- [548] 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. *Journal of Field Robotics*, **29**(5), 762–792.
- [549] Open, NN. 2016. An open source neural networks c++ library. URL: <http://opennn.cimne.com> (2016).
- [550] 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.
- [551] 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.
- [552] Pan, Y., Xiao, P., He, Y., Shao, Z., and Li, Z. 2021a. MULLS: Versatile LiDAR SLAM via Multi-metric Linear Least Square. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [553] Pan, Yue, Xiao, Pengchuan, He, Yujie, Shao, Zhenlei, and Li, Zesong. 2021b. MULLS: Versatile LiDAR SLAM via multi-metric linear least square. Pages 11633–11640 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [554] 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)*.

- [555] 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.
- [556] 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)*.
- [557] Park, Yeong Sang, Shin, Young-Sik, and Kim, Ayoung. 2020. PhaRaO: Direct Radar Odometry using Phase Correlation. Pages 2617–2623 of: *2020 IEEE International Conference on Robotics and Automation (ICRA)*.
- [558] Paskin, M.A. 2003. Thin Junction Tree Filters for Simultaneous Localization and Mapping. In: *Intl. Joint Conf. on AI (IJCAI)*.
- [559] 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**.
- [560] 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.
- [561] 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.
- [562] 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)*.
- [563] Peng, Liangzu, Kümmeler, Christian, and Vidal, René. 2023. 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)*.
- [564] 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*.
- [565] Peng, Songyou, Niemeyer, Michael, Mescheder, Lars, Pollefeys, Marc, and Geiger, Andreas. 2020. Convolutional occupancy networks. In: *European Conf. on Computer Vision (ECCV)*.
- [566] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2024c (May). Quantized Visual-Inertial Odometry. In: *Proc. International Conference on Robotics and Automation*.
- [567] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2024d (May). Ultra-fast Square-Root Filter-based VINS. In: *Proc. International Conference on Robotics and Automation*.
- [568] Peng, Yuxiang, Chen, Chuchu, and Huang, Guoquan. 2025 (May). QVIO2: Quantized MAP-based Visual-Inertial Odometry. In: *Proc. International Conference on Robotics and Automation*.
- [569] 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*. Springer.

- [570] 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)*.
- [571] 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.
- [572] Pineda, Luis, Fan, Taosha, Monge, Maurizio, Venkataraman, Shobha, Sodhi, Paloma, Chen, Ricky T. Q., Ortiz, Joseph, DeTone, Daniel, Wang, Austin S., Anderson, Stuart, Dong, Jing, Amos, Brandon, and Mukadam, Mustafa. 2022a (Oct.). Theseus: A Library for Differentiable Nonlinear Optimization. In: *Conf. Neural Information Processing Systems (NIPS)*.
- [573] Pineda, Luis, Fan, Taosha, Monge, Maurizio, Venkataraman, Shobha, Sodhi, Paloma, Chen, Ricky TQ, Ortiz, Joseph, DeTone, Daniel, Wang, Austin, Anderson, Stuart, Dong, Jing, Amos, Brandon, and Mukadam, Mustafa. 2022b. Theseus: A Library for Differentiable Nonlinear Optimization. *Advances in Neural Information Processing Systems*.
- [574] 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*.
- [575] Pizzoli, Matia, Forster, Christian, and Scaramuzza, Davide. 2014. REMODE: Probabilistic, monocular dense reconstruction in real time. Pages 2609–2616 of: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE.
- [576] Posch, Christoph, Matolin, Daniel, and Wohlgemann, 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.
- [577] Posch, Christoph, Serrano-Gotarredona, Teresa, Linares-Barranco, Bernabe, and Delbruck, Tobi. 2014. Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output. *Proc. IEEE*, **102**(10), 1470–1484.
- [578] 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.
- [579] 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.
- [580] 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)*.
- [581] Qin, T., Li, P., and Shen, S. 2018a. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, **34**(4), 1004–1020.
- [582] Qin, Tong, Li, Peiliang, and Shen, Shaojie. 2018b. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Trans. Robotics*, **34**(4), 1004–1020.
- [583] Qin, Tong, Li, Peiliang, and Shen, Shaojie. 2018c. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Trans. Robotics*, **34**(4), 1004–1020.

- [584] Qiu, Yuheng, Wang, Chen, Xu, Can, Chen, Yutian, Zhou, Xunfei, Xia, Youjie, and Scherer, Sebastian. 2024 (May). *AirIMU: Learning Uncertainty Propagation for Inertial Odometry*.
- [585] Rahimi, Ali, and Recht, Benjamin. 2007. Random features for large-scale kernel machines. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 20.
- [586] Raibert, Marc H. 1986. *Legged robots that balance*. MIT press.
- [587] 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 arXiv:2205.12595*.
- [588] 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.
- [589] Rapp, Matthias, Dietmayer, Klaus, Hahn, Markus, Schuster, Frank, Lombacker, 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)*.
- [590] Rasmussen, Carl Edward, and Williams, Christopher KI. 2006. *Gaussian Processes for Machine Learning*. Cambridge, Mass.: MIT Press.
- [591] Ravichandran, Z., Peng, L., Hughes, N., Griffith, J.D., and Carbone, 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)*. .
- [592] 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.
- [593] Rebecq, Henri, Horstschaefer, Timo, and Scaramuzza, Davide. 2017b. Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Non-linear Optimization. In: *British Machine Vision Conf. (BMVC)*.
- [594] 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.
- [595] 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.
- [596] 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.
- [597] 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*.
- [598] 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)*.

- [599] 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)*.
- [600] 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)*.
- [601] Revels, Jarrett, Lubin, Miles, and Papamarkou, Theodore. 2016. Forward-mode automatic differentiation in Julia. *arXiv preprint arXiv:1607.07892*.
- [602] Robbins, Herbert, and Monro, Sutton. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.
- [603] Rodrigues, Rômulo T., Tsiovas, 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.
- [604] 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*.
- [605] Roriz, Ricardo, Cabral, Jorge, and Gomes, Tiago. 2022a. Automotive LiDAR Technology: A Survey. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, **23**(7), 6282–6297.
- [606] Roriz, Ricardo, Cabral, Jorge, and Gomes, Tiago. 2022b. Automotive LiDAR Technology: A Survey. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, **23**(7), 6282–6297.
- [607] Rosinol, A., Abate, M., Chang, Y., and Carbone, L. 2020a. Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. arXiv preprint: 1910.02490, , , .
- [608] Rosinol, Antoni, Abate, Marcus, Chang, Yun, and Carbone, 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)*.
- [609] Rosinol Vidal, Antoni, Rebecq, Henri, Horstschaefer, 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.
- [610] Rosten, Edward, and Drummond, Tom. 2006. Machine Learning for High-Speed Corner Detection. Pages 430–443 of: *European Conference on Computer Vision (ECCV)*.
- [611] 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.
- [612] 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)*.
- [613] 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)*.
- [614] Rouveure, R., Faure, P., and Monod, M. 2010. Radar-based SLAM without odometric sensor. In: *ROBOTICS2010 : International workshop of Mobile Robotics for environment/agriculture*.

- [615] Rowell, Joseph, Zhang, Lintong, and Fallon, Maurice. 2024. LiSTA: Geometric Object-Based Change Detection in Cluttered Environments. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [616] Rublee, Ethan, Rabaud, Vincent, Konolige, Kurt, and Bradski, Gary. 2011a. ORB: An Efficient Alternative to SIFT or SURF. Pages 2564–2571 of: *International Conference on Computer Vision (ICCV)*.
- [617] Rublee, Ethan, Rabaud, Vincent, Konolige, Kurt, and Bradski, Gary. 2011b. ORB: An efficient alternative to SIFT or SURF. Pages 2564–2571 of: *Intl. Conf. on Computer Vision (ICCV)*. Ieee.
- [618] Rueckauer, Bodo, and Delbruck, Tobi. 2016. Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor. *Front. Neurosci.*, **10**(176).
- [619] Rusinkiewicz, S., and Levoy, M. 2001. Efficient variants of the ICP algorithm. In: *Proc. of Intl. Conf. on 3-D Digital Imaging and Modeling*.
- [620] 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.
- [621] 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.
- [622] 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)*.
- [623] San Segundo, Pablo, and Artieda, Jorge. 2015. A novel clique formulation for the visual feature matching problem. *Appl. Intelligence*, **43**(2), 325–342.
- [624] Sandström, Erik, Li, Yue, Van Gool, Luc, and R. Oswald, Martin. 2023. Point-SLAM: Dense Neural Point Cloud-based SLAM. In: *Intl. Conf. on Computer Vision (ICCV)*.
- [625] 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*.
- [626] Sarlin, Paul-Edouard, Cadena, Cesar, Siegwart, Roland, and Dymczyk, Marcin. 2019. From Coarse to Fine: Robust Hierarchical Localization at Large Scale. In: *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*.
- [627] Sarlin, Paul-Edouard, DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. 2020a. SuperGlue: Learning Feature Matching with Graph Neural Networks. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [628] Sarlin, Paul-Edouard, DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. 2020b. SuperGlue: Learning Feature Matching with Graph Neural Networks. In: *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*.
- [629] Schonberger, Johannes L, and Frahm, Jan-Michael. 2016a. Structure-from-motion revisited. Pages 4104–4113 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

- [630] Schonberger, Johannes L, and Frahm, Jan-Michael. 2016b. Structure-from-motion revisited. Pages 4104–4113 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [631] Schouten, Girmi, and Steckel, Jan. 2017. RadarSLAM: Biomimetic SLAM using ultra-wideband pulse-echo radar. Pages 1–8 of: *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*.
- [632] Schubert, D., Demmel, N., Usenko, V., Stueckler, J., and Cremers, D. 2018 (September). Direct Sparse Odometry With Rolling Shutter. In: *European Conference on Computer Vision (ECCV)*.
- [633] Schubert, D., Demmel, N., von Stumberg, L., Usenko, V., and Cremers, D. 2019 (November). Rolling-Shutter Modelling for Visual-Inertial Odometry. In: *International Conference on Intelligent Robots and Systems (IROS)*.
- [634] 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)*.
- [635] 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.
- [636] Segal, Aleksandr, Haehnel, Dirk, and Thrun, Sebastian. 2009. Generalized-icp. Page 435 of: *Robotics: Science and Systems (RSS)*, vol. 2. Seattle, WA.
- [637] 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.
- [638] Semini, Claudio, and Wieber, Pierre-Brice. 2020. *Legged Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg. Pages 1–8.
- [639] 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**.
- [640] 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.
- [641] Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., and Rus, D. 2020a. LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [642] 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)*.
- [643] 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.
- [644] Shan, Tixiao, Englot, Brendan, Duarte, Fábio, Ratti, Carlo, and Rus, Daniela. 2021. Robust place recognition using an imaging lidar. Pages 5469–5475 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.

- [645] Sheeny, Marcel, De Pellegrin, Emanuele, Mukherjee, Saptarshi, Ahrabian, 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)*.
- [646] 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.
- [647] 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.
- [648] Shi, J., and Tomasi, C. 1994. Good features to track. Pages 593–600 of: *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*.
- [649] 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, .
- [650] Shi, J., Yang, H., and Carlone, L. 2023a. Optimal and Robust Category-level Perception: Object Pose and Shape Estimation from 2D and 3D Semantic Keypoints. *IEEE Trans. Robotics*, **39**(5), 4131–4151. .
- [651] Shi, Pengcheng, Zhang, Yongjun, and Li, Jiayuan. 2023b. Lidar-based place recognition for autonomous driving: A survey. *arXiv preprint*.
- [652] Shiba, Shintaro, Aoki, Yoshimitsu, and Gallego, Guillermo. 2022. Event Collapse in Contrast Maximization Frameworks. *Sensors*, **22**(14), 1–20.
- [653] 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.
- [654] Siciliano, Bruno, Sciavicco, Lorenzo, Villani, Luigi, and Oriolo, Giuseppe. 2008. *Robotics: Modelling, Planning and Control*. 1st edn. Springer Publishing Company, Incorporated. Chap. 5.
- [655] 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: *Proc. of the 3rd Canadian Conf. on Computer and Robotic Vision (CRV)*.
- [656] Smith, R., and Cheeseman, P. 1987. On the representation and estimation of spatial uncertainty. *Intl. J. of Robotics Research*, **5**(4), 56–68.
- [657] 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.
- [658] Sola, Joan, Deray, Jeremie, and Atchuthan, Dinesh. 2018. A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*.
- [659] 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)*.
- [660] Stadie, Bradly, Zhang, Lunjun, and Ba, Jimmy. 2020. Learning intrinsic rewards as a bi-level optimization problem. Pages 111–120 of: *Conference on Uncertainty in Artificial Intelligence*. PMLR.

- [661] Stathoulopoulos, 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.
- [662] 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)*.
- [663] Steinbruecker, F., Kerl, C., Sturm, J., and Cremers, D. 2013. Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences. In: *IEEE International Conference on Computer Vision (ICCV)*.
- [664] Steinbruecker, F., Sturm, J., and Cremers, D. 2014. Volumetric 3D Mapping in Real-Time on a CPU. In: *International Conference on Robotics and Automation (ICRA)*.
- [665] Stillwell, J. 2008. *Naive Lie Theory*. Springer.
- [666] 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)*.
- [667] 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)*.
- [668] 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)*.
- [669] Strasdat, H., Montiel, Jose M. M., and Davison, A. J. 2012. Visual SLAM: Why filter? *Image and Vision Computing*, **30**(2), 65–77.
- [670] 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.
- [671] 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)*.
- [672] Stumberg, Lukas von, and Cremers, Daniel. 2022. DM-VIO: Delayed Marginalization Visual-Inertial Odometry. *IEEE Robotics and Automation Letters*, **7**(2), 1408–1415.
- [673] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. 2012 (Oct.). A Benchmark for the Evaluation of RGB-D SLAM Systems. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [674] 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)*.
- [675] 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)*.
- [676] 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)*.
- [677] 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)*. , , highlighted in the MIT News: other media coverage: .
- [678] Sun, Scott, Melamed, Dennis, and Kitani, Kris. 2021. IDOL: Inertial deep orientation-estimation and localization. Pages 6128–6137 of: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35.
- [679] Sun, Shuo, Mielle, Malcolm, Lilienthal, Achim J., and Magnusson, Martin. 2024. 3QFP: Efficient neural implicit surface reconstruction using Tri-Quadtrees and Fourier feature Positional encoding. Pages 4036–4044 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [680] Sünderhauf, N., and Protzel, P. 2012a. Switchable Constraints for Robust Pose Graph SLAM. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [681] Sünderhauf, N., and Protzel, P. 2012b. Towards a robust back-end for pose graph SLAM. Pages 1254–1261 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.
- [682] Sünderhauf, Niko, and Protzel, Peter. 2012. Switchable Constraints for Robust Pose Graph SLAM. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [683] Sünderhauf, Niko, and Protzel, Peter. 2013. Switchable Constraints vs. Max-Mixture Models vs. RRR – A Comparison of three Approaches to Robust Pose Graph SLAM. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [684] Takeuchi, Eijiyo, Elfes, Alberto, and Roberts, Jonathan. 2015. *Localization and Place Recognition Using an Ultra-Wide Band (UWB) Radar*. Cham: Springer International Publishing. Pages 275–288.
- [685] Tang, Chengzhou, and Tan, Ping. 2019a. Ba-net: Dense bundle adjustment network. *International Conference on Learning Representations (ICLR)*.
- [686] Tang, Chengzhou, and Tan, Ping. 2019b. BA-Net: Dense bundle adjustment networks. *7th International Conference on Learning Representations, ICLR 2019*.
- [687] 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.
- [688] 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.
- [689] 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. Computer and Robot Vision*. IEEE.
- [690] Tedrake, Russ, and the Drake Development Team. 2019. *Drake: Model-based design and verification for robotics*.
- [691] Teed, Zachary, and Deng, Jia. 2018. Deepv2d: Video to depth with differentiable structure from motion. *arXiv preprint arXiv:1812.04605*.
- [692] Teed, Zachary, and Deng, Jia. 2021a. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems*, **34**.
- [693] 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)*.
- [694] Teed, Zachary, Lipson, Lahav, and Deng, Jia. 2023. Deep Patch Visual Odometry. In: *Advances in Neural Information Processing Systems (NIPS)*.
- [695] 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)*.
- [696] 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.
- [697] Thrun, S., Burgard, W., and Fox, D. 2005. *Probabilistic Robotics*. The MIT press, Cambridge, MA.
- [698] Thrun, Sebastian, et al. 2002. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, **1**(1-35), 1.
- [699] Tian, Y., Chang, Y., Arias, F., Herrera, Nieto-Granda, C., How, J.P., and Carbone, L. 2022a. Kimera-Multi: Robust, Distributed, Dense Metric-Semantic SLAM for Multi-Robot Systems. *IEEE Trans. Robotics*. accepted, arXiv preprint: 2106.14386, .
- [700] Tian, Yulun, Chang, Yun, Herrera Arias, Fernando, Nieto-Granda, Carlos, How, Jonathan P., and Carbone, Luca. 2022b. Kimera-Multi: Robust, Distributed, Dense Metric-Semantic SLAM for Multi-Robot Systems. *IEEE Trans. Robotics*, **38**(4), 2022–2038.
- [701] 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.
- [702] Titterton, D., and Weston, J. 2005. *Strapdown Inertial Navigation Technology*. second edn. The Institution of Engineering and Technology.
- [703] Todd, D. J. 1985. *A brief history of walking machines*. Boston, MA: Springer US. Pages 169–177.
- [704] Tokui, Seiya, Oono, Kenta, Hido, Shohei, and Clayton, Justin. 2015. Chainer: a next-generation open source framework for deep learning. Pages 1–6 of: *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, vol. 5.
- [705] 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.
- [706] 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.
- [707] Trawny, N., and Roumeliotis, S.I. 2005. Indirect Kalman Filter for 3D Attitude Estimation. *Mars Lab, Technical Report Number 2005-002, Rev. 57*.
- [708] 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)*.
- [709] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. 2000a. Bundle Adjustment – A Modern Synthesis. Pages 298–372 of: Triggs, W., Zisserman,

- A., and Szeliski, R. (eds), *Vision Algorithms: Theory and Practice*. LNCS, vol. 1883. Springer Verlag.
- [710] Triggs, Bill, McLauchlan, Philip F, Hartley, Richard I, and Fitzgibbon, Andrew W. 1999. Bundle adjustment—a modern synthesis. Pages 298–372 of: *International workshop on vision algorithms*. Springer.
- [711] Triggs, Bill, McLauchlan, Philip F, Hartley, Richard I, and Fitzgibbon, Andrew W. 2000b. 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.
- [712] 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.
- [713] 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.
- [714] Tseng, Paul. 2001. Convergence of a block coordinate descent method for non-differentiable minimization. *Journal of optimization theory and applications*, **109**, 475–494.
- [715] 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.
- [716] Usenko, V., Engel, J., Stückler, J., and Cremers, D. 2016a (May). Direct Visual-Inertial Odometry with Stereo Cameras. Pages 1885–1892 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [717] Usenko, V., Demmel, N., Schubert, D., Stueckler, J., and Cremers, D. 2020. Visual-Inertial Mapping with Non-Linear Factor Recovery. *IEEE Robotics and Automation Letters (RA-L) and Int. Conference on Intelligent Robotics and Automation (ICRA)*, **5**(2), 422–429.
- [718] Usenko, Vladyslav, Engel, Jakob, Stückler, Jörg, and Cremers, Daniel. 2016b (May). Direct visual-inertial odometry with stereo cameras. Pages 1885–1892 of: *IEEE International Conference on Robotics and Automation*.
- [719] Usenko, Vladyslav, Demmel, Nikolaus, and Cremers, Daniel. 2018. The double sphere camera model. Pages 552–560 of: *2018 International Conference on 3D Vision (3DV)*. IEEE.
- [720] Usenko, Vladyslav, 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.
- [721] 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)*.
- [722] 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.
- [723] 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)*.
- [724] 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.
- [725] Vigne, Matthieu, Khoury, 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.
- [726] 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)*.
- [727] Vizzo, Ignacio, Guadagnino, Tiziano, Behley, Jens, and Stachniss, Cyrill. 2022. VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data. *IEEE Sensors*, **22**(3).
- [728] 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.
- [729] von Stumberg, L., and Cremers, D. 2022. DM-VIO: Delayed Marginalization Visual-Inertial Odometry. *IEEE Robotics and Automation Letters (RA-L) and International Conference on Robotics and Automation (ICRA)*, **7**(2), 1408–1415.
- [730] von Stumberg, L., Usenko, V., and Cremers, D. 2018 (May). Direct Sparse Visual-Inertial Odometry using Dynamic Marginalization. In: *International Conference on Robotics and Automation (ICRA)*.
- [731] 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.
- [732] 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, Carloni, Luca, Hutter, Marco, and Scherer, Sebastian. 2023. PyPose: A Library for Robot Learning with Physics-based Optimization. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [733] 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. 2024. Imperative Learning: A Self-supervised Neural-Symbolic Learning Framework for Robot Autonomy. *arXiv preprint*.
- [734] 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)*.

- [735] 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.
- [736] Wang, Han, Wang, Chen, and Xie, Lihua. 2020b. 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.
- [737] 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.
- [738] 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)*.
- [739] Wang, K., Gao, F., and Shen, S. 2019. Real-Time Scalable Dense Surfel Mapping. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [740] 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)*.
- [741] 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.
- [742] Wang, Wenshan, Zhu, Delong, Wang, Xiangwei, Hu, Yaoyu, Qiu, Yuheng, Wang, Chen, Hu, Yafei, Kapoor, Ashish, and Scherer, Sebastian. 2020c. Tartanair: 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.
- [743] Wang, Wenshan, Hu, Yaoyu, and Scherer, Sebastian. 2021b. Tartanvo: A generalizable learning-based vo. Pages 1761–1772 of: *Conference on Robot Learning*. PMLR.
- [744] Weber, S., Demmel, N., and Cremers, D. 2021. Multidirectional Conjugate Gradients for Scalable Bundle Adjustment. In: *German Conference on Pattern Recognition (GCPR)*.
- [745] Weber, S., Demmel, N., Chan, T Chon, and Cremers, D. 2023a. Power Bundle Adjustment for Large-Scale 3D Reconstruction. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [746] 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)*.
- [747] Weber, Simon, Demmel, Nikolaus, Chan, Tin Chon, and Cremers, Daniel. 2023b. Power bundle adjustment for large-scale 3d reconstruction. Pages 281–289 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [748] Wei, Peng, Hua, Guoliang, Huang, Weibo, Meng, Fanyang, and Liu, Hong. 2021. Unsupervised monocular visual-inertial odometry network. Pages 2347–2354 of: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*.
- [749] Wei, T., Patel, Y., Shekhovtsov, A., Matas, J., and Barath, D. 2023. Generalized differentiable RANSAC. In: *Intl. Conf. on Computer Vision (ICCV)*.

- [750] 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)*.
- [751] 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)*.
- [752] Weitkamp, Claus. 2006. *Lidar: range-resolved optical remote sensing of the atmosphere*. Vol. 102. Springer Verlag.
- [753] Wendel, Andreas, Maurer, Michael, Graber, Gottfried, Pock, Thomas, and Bischof, Horst. 2012. Dense reconstruction on-the-fly. Pages 1450–1457 of: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.
- [754] 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: *Proceedings of the German Conference on Pattern Recognition (GCPR)*.
- [755] Whelan, T., Leutenegger, S., Moreno, R. S., Glocker, B., and Davison, A. 2015a. ElasticFusion: Dense SLAM Without A Pose Graph. In: *Robotics: Science and Systems (RSS)*.
- [756] Whelan, Thomas, Kaess, Michael, Fallon, Maurice, Johannsson, Hordur, Leonard, John, and McDonald, John. 2012. Kintinuous: Spatially extended kinectfusion.
- [757] Whelan, Thomas, Leutenegger, Stefan, Salas-Moreno, Renato F., Glocker, Ben, and Davison, Andrew J. 2015b. ElasticFusion: Dense SLAM Without A Pose Graph. In: *Robotics: Science and Systems (RSS)*.
- [758] 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.
- [759] Williams, Oliver, and Fitzgibbon, Andrew. 2007. *Gaussian Process Implicit Surfaces*.
- [760] 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)*.
- [761] Wisth, David, Camurri, Marco, and Fallon, Maurice. 2022. VILENS: Visual, Inertial, Lidar, and Leg Odometry for All-Terrain Legged Robots. *IEEE Trans. Robotics*, Aug., 1–18.
- [762] Wisth, David, Camurri, Marco, and Fallon, Maurice. 2023. VILENS: Visual, Inertial, Lidar, and Leg Odometry for All-Terrain Legged Robots. *IEEE Trans. Robotics*, **39**(1), 309–326.
- [763] 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)*.
- [764] Wu, Lan, Lee, Ki Myung Brian, Liu, Liyang, and Vidal-Calleja, Teresa. 2021. Faithful Euclidean distance field from log-Gaussian process implicit surfaces. *IEEE Robotics and Automation Letters*, 2461–2468.
- [765] 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.

- [766] Wu, Qinghua, and Hao, Jin-Kao. 2015. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, **242**(3), 693–709.
- [767] Wu, Xiaoyi, Chen, Yushuai, Li, Zhan, Hong, Ziyang, and Hu, Liang. 2024. EFEAR-4D: Ego-Velocity Filtering for Efficient and Accurate 4D Radar Odometry. *IEEE Robotics and Automation Letters*, **9**(11), 9828–9835.
- [768] 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.
- [769] 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)*.
- [770] 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)*.
- [771] 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)*.
- [772] 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)*.
- [773] Xu, R., Dong, W., Sharma, A., and Kaess, M. 2022a (Oct.). Learned Depth Estimation of 3D Imaging Radar for Indoor Mapping. In: *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [774] Xu, Wei, Cai, Yixi, He, Dongjiao, Lin, Jiarong, and Zhang, Fu. 2022b. Fastlio2: Fast direct lidar-inertial odometry. *IEEE Trans. Robotics*, **38**(4), 2053–2073.
- [775] Xu, Wei, Cai, Yixi, He, Dongjiao, Lin, Jiarong, and Zhang, Fu. 2022c. Fastlio2: Fast direct lidar-inertial odometry. *IEEE Trans. Robotics*, **38**(4), 2053–2073.
- [776] 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*.
- [777] Yan, Hang, Shan, Qi, and Furukawa, Yasutaka. 2018. RIDI: Robust IMU double integration. Pages 621–636 of: *Proceedings of the European conference on computer vision (ECCV)*.
- [778] Yang, Fan, Wang, Chen, Cadena, Cesar, and Hutter, Marco. 2023a. iPlanner: Imperative Path Planning. In: *Robotics: Science and Systems (RSS)*.
- [779] Yang, H., and Carlone, L. 2019a. A Polynomial-time Solution for Robust Registration with Extreme Outlier Rates. In: *Robotics: Science and Systems (RSS)*. , , , .
- [780] 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)*. (Oral Presentation, accept rate: 4%), Arxiv version: 1905.12536, .
- [781] 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. .

- [782] Yang, H., and Carlone, L. 2022. Certifiably Optimal Outlier-Robust Geometric Perception: Semidefinite Relaxations and Scalable Global Optimization. *IEEE Trans. Pattern Anal. Machine Intell.* .
- [783] Yang, H., Antonante, P., Tzoumas, V., and Carlone, L. 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. arXiv preprint:1909.08605 (with supplemental material), .
- [784] Yang, H., Shi, J., and Carlone, L. 2020b. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. Robotics*, **37**(2), 314–333. extended arXiv version 2001.07715 .
- [785] Yang, Heng, and Carlone, Luca. 2023. Certifiably Optimal Outlier-Robust Geometric Perception: Semidefinite Relaxations and Scalable Global Optimization. *IEEE Trans. Pattern Anal. Machine Intell.*, **45**(3), 2816–2834.
- [786] Yang, Heng, Shi, Jingnan, and Carlone, Luca. 2020c. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. Robotics*, **37**(2).
- [787] 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.
- [788] 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.
- [789] Yang, N., Wang, R., Gao, X., and Cremers, D. 2018. 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.
- [790] 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)*.
- [791] Yang, Wen, Gong, Zheng, Huang, Baifu, and Hong, Xiaoping. 2022. Lidar With Velocity: Correcting Moving Objects Point Cloud Distortion From Oscillating Scanning Lidars by Fusion With Camera. *IEEE Robotics and Automation Letters*, **7**(3).
- [792] Yang, Yulin. 2024. *Aided Inertial Navigation System: Analysis and Algorithms*. phdthesis, Univiersity of Delaware.
- [793] Yang, Yulin, and Huang, Guoquan. 2019. Observability Analysis of Aided INS with Heterogeneous Features of Points, Lines and Planes. *IEEE Transactions on Robotics*, **35**(6), 399–1418.
- [794] 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 (RA-L)*, **4**(2), 2070–2077.
- [795] 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: *Proc. International Conference on Robotics and Automation*.
- [796] Yang, Yulin, Geneva, Patrick, Zuo, Xingxing, and Huang, Guoquan. 2023c. Online Self-Calibration for Visual-Inertial Navigation Systems: Models, Analysis and Degeneracy. *IEEE Transactions on Robotics*, May.
- [797] 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.
- [798] Yannakakis, M. 1981. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, **2**.
- [799] 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)*.
- [800] Yi, Brent, Lee, Michelle A, Kloss, Alina, Martín-Martín, Roberto, and Bohg, Jeannette. 2021a. Differentiable factor graph optimization for learning smoothers. Pages 1339–1345 of: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [801] Yi, Brent, Lee, Michelle, Kloss, Alina, Martín-Martín, Roberto, and Bohg, Jeannette. 2021b. Differentiable Factor Graph Optimization for Learning Smoothers. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.
- [802] Yi, Kwang Moo, Verdie, Yannick, Lepetit, Vincent, and Fua, Pascal. 2016. LIFT: Learned Invariant Feature Transform. In: *European Conference on Computer Vision (ECCV)*.
- [803] 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**.
- [804] 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.
- [805] Yin, Huan, Chen, Runjian, Wang, Yue, and Xiong, Rong. 2022a. RaLL: End-to-End Radar Localization on Lidar Map Using Differentiable Measurement Model. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, **23**(7), 6737–6750.
- [806] 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.
- [807] Yin, Jie, Li, Ang, Li, Tao, Yu, Wenxian, and Zou, Danping. 2022b. M2DGR: A Multi-Sensor and Multi-Scenario SLAM Dataset for Ground Robots. *IEEE Robotics and Automation Letters*, **7**(2), 2266–2273.
- [808] 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*.
- [809] 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)*.
- [810] 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)*.
- [811] 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*.
- [812] 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.
- [813] 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.
- [814] Yuan, Wenzhen, and Ramalingam, Srikumar. 2016. Fast Localization and Tracking using Event Sensors. Pages 4564–4571 of: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [815] Yugay, Vladimir, Li, Yue, Gevers, Theo, and Oswald, Martin R. 2023. Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting. *arXiv preprint*.
- [816] Zhan, Zitong, Gao, Dasong, Lin, Yun-Jou, Xia, Youjie, and Wang, Chen. 2024. iMatching: Imperative Correspondence Learning. In: *European Conference on Computer Vision (ECCV)*.
- [817] Zhang, J., and Singh, S. 2014. LOAM: Lidar Odometry and Mapping in Real-time. In: *Robotics: Science and Systems (RSS)*.
- [818] 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)*.
- [819] 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)*.
- [820] 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.
- [821] 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)*.
- [822] Zhang, Ming, Zhang, Mingming, Chen, Yiming, and Li, Mingyang. 2021. IMU Data Processing For Inertial Aided Navigation: A Recurrent Neural Network Based Approach. Pages 3992–3998 of: *2021 IEEE International Conference on Robotics and Automation (ICRA)*.
- [823] Zhang, Z., Suleiman, A., Carlone, L., Sze, V., and Karaman, S. 2017. Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach. In: *Robotics: Science and Systems (RSS)*. , highlighted in the MIT News: .
- [824] Zhang, Zelin, Yezzi, Anthony, and Gallego, Guillermo. 2023e. 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.
- [825] Zhang, Zhengyou. 1994. Iterative point matching for registration of free-form curves and surfaces. *Intl. J. of Computer Vision*, **13**, 119–152.
- [826] 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)*.
- [827] 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)*.
- [828] Zhao, Shibo, Wang, Peng, Zhang, Hengrui, Fang, Zheng, and Scherer, Sebastian. 2020. 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.
- [829] Zhao, Shibo, Zhang, Hengrui, Wang, Peng, Nogueira, Lucas, and Scherer, Sebastian. 2021a. Super odometry: IMU-centric LiDAR-visual-inertial estimator for challenging environments. Pages 8729–8736 of: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- [830] Zhao, Shibo, Zhang, Hengrui, Wang, Peng, Nogueira, Lucas, and Scherer, Sebastian. 2021b. 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.
- [831] 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*.
- [832] 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)*.
- [833] Zhong, Xinguang, Pan, Yue, Behley, Jens, and Stachniss, Cyrill. 2023a. SHINE-Mapping: Large-Scale 3D Mapping Using Sparse Hierarchical Implicit Neural Representations. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.
- [834] Zhong, Xinguang, Pan, Yue, Behley, Jens, and Stachniss, Cyrill. 2023b. Shine-mapping: Large-scale 3d mapping using sparse hierarchical implicit neural representations. Pages 8371–8377 of: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- [835] Zhou, Kun, Hou, Qiming, Wang, Rui, and Guo, Baining. 2008. Real-time kd-tree construction on graphics hardware. *ACM Transactions on Graphics (TOG)*, **27**(5), 1–11.
- [836] Zhou, Q.Y., Park, J., and Koltun, V. 2016. Fast global registration. Pages 766–782 of: *European Conf. on Computer Vision (ECCV)*. Springer.
- [837] Zhou, Shenghao, Katragadda, Saimouli, and Huang, Guoquan. 2025 (May). Learning IMU Bias with Diffusion Model. In: *Proc. International Conference on Robotics and Automation*.

- [838] Zhou, Yi, Gallego, Guillermo, Rebecq, Henri, Kneip, Laurent, Li, Hong-dong, and Scaramuzza, Davide. 2018. Semi-Dense 3D Reconstruction with a Stereo Event Camera. Pages 242–258 of: *European Conf. on Computer Vision (ECCV)*.
- [839] Zhou, Yi, Gallego, Guillermo, and Shen, Shaojie. 2021. Event-based Stereo Visual Odometry. *IEEE Trans. Robotics*, **37**(5), 1433–1450.
- [840] 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)*.
- [841] 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.
- [842] Zhu, Alex Zihao, Yuan, Liangzhe, Chaney, Kenneth, and Daniilidis, Kostas. 2019. Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion. Pages 989–997 of: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [843] 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)*.
- [844] 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)*.
- [845] 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.
- [846] Zhuang, Ziwen, Fu, Zipeng, Wang, Jianren, Atkeson, Christopher G., Schwerfeger, 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.
- [847] 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)*.
- [848] Zuo, X., Xie, J., Liu, Y., and Huang, Guoquan. 2017 (Sept.). Robust Visual SLAM with Point and Line Features. Pages 1775–1782 of: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- [849] Zuo, Yi-Fan, Xu, Wanting, Wang, Xia, Wang, Yifu, and Kneip, Laurent. 2024. Cross-Modal Semidense 6-DOF Tracking of an Event Camera in Challenging Conditions. *IEEE Trans. Robotics*, **40**, 1600–1616.
- [850] Zwicker, Matthias, Pfister, Hanspeter, van Baar, Jeroen, and Gross, Markus. 2001. Surface Splatting. In: *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- [851] 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

## Subject index