

Multiple-robot Simultaneous Localization and Mapping

by

Sajad Saeedi

M.Sc.E., Tarbiat Modares University, 2004

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree
of

Doctor of Philosophy

In the Graduate Academic Unit of Electrical and Computer Engineering

Supervisor(s): Howard Li, Ph.D., P.Eng

Examining Board: John Kershaw, Ph.D., Jr. RPF, CF,
Acting Associate Dean of Graduate Studies, Chair
Brent Petersen, Ph.D., P.Eng, Electrical and Computer Engineering
Julian Meng, Ph.D., P.Eng, Electrical and Computer Engineering
Juan Carretero, Ph.D., P.Eng, Mechanical Engineering

External Examiner: Hugh H.T. Liu, Ph.D., P.Eng, F.CSME, AF.AIAA,
University of Toronto Institute of Aerospace Studies

**This dissertation is accepted by
Dean of Graduate Studies**

THE UNIVERSITY OF NEW BRUNSWICK

February, 2014

©Sajad Saeedi, 2014

Abstract

Mobile robotic systems are developing very rapidly; however, there are key requirements in real-world, and especially GPS-denied applications, which without them no autonomy can be achieved. One such requirement is the simultaneous localization and mapping (SLAM). SLAM is a process in which an unknown environment is explored and mapped consistently. The robot placed in an *a priori* unknown environment builds a map of the environment and also situates itself within the map simultaneously. The focus of this research is 1) achieving SLAM with a team of ground robots and 2) performing SLAM and autonomous navigation by a quadrotor rotorcraft.

Many problems exist in multiple-robot SLAM, including finding the relative poses of the robots, accounting for the uncertainty of the relative poses, fusing maps, dealing with complexity issues, updating maps and poses in the global perspective, and many more. In this thesis, two general approaches are taken to tackle these problems. The first is map merging, which is based on sharing processed data, such as the local maps developed by the robots. The second is a particle filter which is primarily based on sharing and processing raw data.

In map merging, four different techniques are proposed: 1) map segmentation, 2) neural network self-organizing maps, 3) probabilistic generalized Voronoi diagram, and 4) Hough peak matching. In map segmentation, unique geometric features from maps are selected and compared with one another to find the required transformation between the poses of the robots. In the neural network approach, maps are processed

by self-organizing maps to find the best alignment. In the method of the probabilistic generalized Voronoi diagram, probabilistic skeletons of maps are extracted and then cross matched with each other to find the relative rotation and the translation. Finally, in Hough peak matching, all maps are transformed to the Hough space and a solution is proposed to find the transformation in this space. The Hough space has interesting properties that make this solution robust and efficient. In particle filter-based multiple-robot SLAM, a hybrid solution is proposed that uses map merging to find the relative poses and then proceeds with a particle filter to process all raw data received from all robots.

In another part of the thesis, SLAM and autonomous navigation are investigated for quadrotors. Unlike ground robots, quadrotors have highly nonlinear and complicated motions. Quadrotors require special consideration when dealing with perception and navigation. A complete autonomy package is proposed which enables quadrotors to fly autonomously and perform SLAM in GPS-denied environments.

For each of the proposed solutions, experimental results from known datasets, simulation environments, and real-world experiments are presented.

Acknowledgements

I would like to thank my supervisor, Dr. Howard Li, for his excellent support, expertise, motivation, and encouragement to move things forward. He granted me the freedom that I needed in developing my own ideas and at the same time helped me to balance all I needed to make things happen.

I would also like to thank the review committee for taking the time to review the thesis and provide feedback.

The advice that I have received from Dr. Brent Petersen, Dr. Kevin Englehart, and Dr. Chris Diduch, among others, is highly appreciated.

Being a member of the COBRA group has given me the opportunity to meet and work with excellent people. In particular, I am indebted to Liam Paull for his collaboration and fruitful discussions. Thanks to his mom, Jacqueline Paull, retired math teacher, for her help with the topological mapping formulation. Many thanks also go to Amr Nagaty for brainstorming with me and providing feedback on my work, to Carl Thibault without whom I would not be able to fly robots and perform experiments, and to Yao Hong Kok and Shang Yang who helped me with the hardware setup.

I also would like to thank Shelley Cormier, Karen Annett, Denise Burke, Ryan Jennings, Michael Morton, Troy Lavigne, and John Landry for their administrative and technical support during my projects.

I owe my deepest gratitude and thanks to my family, and especially my wife Anita, for their support and love during my studies at the university.

This work has been supported by the National Science and Engineering Research Council (NSERC), Defense Research and Development Canada (DRDC), Canada Foundation for Innovation (CFI), and New Brunswick Innovation Foundation (NBIF). Their support is gratefully acknowledged.

Contents

Abstract	ii
Acknowledgments	iv
Table of Contents	vi
List of Tables	xiv
List of Figures	xviii
Abbreviations	xix
Preface	xxi
I Introduction	1
1 Multiple-robot SLAM	5
1.1 Problem Statement	5
1.1.1 Relative Poses of Robots	6
1.1.2 Uncertainty of the Relative Poses	6
1.1.3 Updating Maps and Poses	7
1.1.4 Line-of-sight Observations	7
1.1.5 Closing Loops	7

1.1.6	Complexity	8
1.1.7	Communications	8
1.2	Significance and Objectives	8
1.3	Literature Review	9
1.3.1	Relative Poses of Robots	11
1.3.2	Uncertainty of the Relative Poses	12
1.3.3	Updating Maps and Poses	12
1.3.4	Line-of-sight Observations	13
1.3.5	Closing Loops	13
1.3.6	Complexity	13
1.3.7	Communications	13
1.4	Proposed Method	14
1.4.1	Relative Poses of Robots	14
1.4.2	Uncertainty of the Relative Poses	15
1.4.3	Updating Maps and Poses	16
1.4.4	Line-of-sight Observations	16
1.4.5	Closing Loops	16
1.4.6	Complexity	17
1.4.7	Communications	17
1.5	Contributions	17
2	Rotorcraft SLAM	20
2.1	Problem Statement	20
2.1.1	Nonlinear and Fast Dynamics and Vibration Effects	20
2.1.2	Limited Payload	21
2.1.3	Odometry Limitation	21
2.1.4	Perception	22
2.1.5	Navigation	22

2.2	Significance and Objectives	23
2.3	Literature Review	23
2.3.1	Localization and Mapping	24
2.3.2	Path Planning	26
2.4	Proposed Method	27
2.5	Contributions	29
II	Multiple-robot SLAM	31
3	Background	33
3.1	Sensing Devices	33
3.1.1	Scanning Laser Ranger	34
3.1.2	Sound Detection and Ranging	34
3.1.3	Optical Sensors	34
3.1.4	Magnetometer	35
3.1.5	Inertial Sensors	35
3.1.6	Wheel Encoder	36
3.1.7	Radio-frequency Identification	37
3.1.8	Global Positioning System	37
3.1.9	Motion Capture System	37
3.2	Probabilistic Estimation	38
3.2.1	Conditional Probability	38
3.2.2	Bayes Rule	40
3.2.3	Marginalization	41
3.2.4	Probabilistic Models	41
3.2.5	Bayes Filter	42
3.2.6	Kalman Filter	44
3.2.7	Extended Kalman Filter	45

3.2.8	Information Filter	47
3.2.9	Particle Filter	48
3.2.10	Least Squares Regression	53
3.3	Background on Single-robot SLAM	54
3.3.1	Localization	54
3.3.2	Mapping	56
3.3.3	Definition of Single-robot SLAM	60
3.3.4	Filtering Solutions for SLAM	64
3.3.5	Smoothing Solutions for SLAM	69
3.3.6	Bio-inspired Solutions	73
3.3.7	2D and 3D SLAM	73
3.4	Background on Multiple-robot SLAM	75
3.4.1	Multi-agent Systems	75
3.4.2	Current Solutions for Multiple-robot SLAM	79
3.5	Summary	85
4	Map Merging	86
4.1	System Overview	86
4.1.1	Problem Statement	86
4.1.2	Problems with Existing Methods and Motivations	88
4.1.3	Description of the Proposed Methods for Map Merging	88
4.1.4	Advantages and Disadvantages	89
4.1.5	Summary	90
4.2	Map Segmentation	90
4.2.1	Problems with Existing Methods and Motivations	90
4.2.2	Background: Radon Transform	91
4.2.3	Description of the Method	91
4.2.4	Advantages and Disadvantages	106

4.2.5	Contribution	106
4.2.6	Experiment	107
4.2.7	Summary	121
4.3	Neural Networks Approach	122
4.3.1	Problems with Existing Methods and Motivations	122
4.3.2	Background: Self-Organizing Maps	123
4.3.3	Description of the Proposed Method	124
4.3.4	Advantages and Disadvantages	135
4.3.5	Discussion	135
4.3.6	Contribution	136
4.3.7	Experiment	137
4.3.8	Summary	147
4.4	Probabilistic Generalized Voronoi Diagram	148
4.4.1	Problems with Existing Methods and Motivations	148
4.4.2	Definition	150
4.4.3	Description of the Method	152
4.4.4	Advantages and Disadvantages	171
4.4.5	Contribution	172
4.4.6	Experiment	172
4.4.7	Summary	178
4.5	Hough Peak Matching	178
4.5.1	Problems with Existing Methods and Motivations	179
4.5.2	Background: Hough Transform	180
4.5.3	Description of the Method	182
4.5.4	Advantages and Disadvantages	194
4.5.5	Contribution	194
4.5.6	Experiment	194

4.5.7	Summary	203
4.6	Summary of Map Merging	204
5	Particle Filter-based Multiple-robot SLAM	207
5.1	System Overview	207
5.1.1	Problem Statement	208
5.1.2	Description of the Proposed Method	209
5.1.3	Summary	210
5.2	Multiple-robot SLAM	210
5.2.1	Problem Statement	210
5.2.2	Problems with Existing Methods and Motivations	211
5.2.3	Notation and Definition	213
5.2.4	Description of the Proposed Method	216
5.2.5	Advantages and Disadvantages	247
5.2.6	Discussion	248
5.2.7	Contribution	252
5.2.8	Experiment	254
5.2.9	Summary	262
III	Rotorcraft SLAM	264
6	Background	266
6.1	Quadrotor	266
6.1.1	History	266
6.1.2	Motivation	267
6.1.3	Quadrotor Model	269
6.1.4	Advantage and Disadvantage	273
6.2	Quadrotor Navigation	274

6.2.1	Control	275
6.2.2	State Estimation	275
6.2.3	Map Learning	279
6.2.4	Mission Planning	284
6.3	Summary	285
7	Perception, Navigation, and Autonomy	286
7.1	System Overview	286
7.2	Proposed Perception and Autonomy Solution	287
7.2.1	Problem Statement	287
7.2.2	Problems with Existing Methods and Motivations	288
7.2.3	Description of the Method	289
7.2.4	Advantages and Disadvantages	304
7.2.5	Contribution	306
7.3	Experiment	307
7.3.1	Data Collection	307
7.3.2	Case1: Simulation in Gazebo	308
7.3.3	Case2: Unmanned Ground Robot	314
7.3.4	Case3: Quadrotor	315
7.3.5	Discussion	320
7.4	Summary	321
IV	Conclusions	322
8	Multiple-robot SLAM	324
8.1	Summary of Contributions	325
8.1.1	Map Merging	325
8.1.2	Particle Filter-based Multiple-robot SLAM	326

8.2	Future Research	328
9	Rotorcraft SLAM	329
9.1	Summary of Contributions	329
9.2	Future Research	330
	Bibliography	333
	Vita	

List of Tables

3.1	Some general methods for state estimation	55
3.2	Comparison of different mapping methods	58
3.3	Comparison of some common SLAM techniques.	74
4.1	List of important parameters with their nominal values.	119
4.2	Comparison of processing times	120
4.3	Comparison of verification indices	120
4.4	Comparison of processing times	147
4.5	Processing time and efficiency of three experiments.	178
4.6	Percentage of approximate overlaps between maps.	199
4.7	List of parameters with their nominal values.	201
4.8	Processing time and efficiency of three experiments.	202
5.1	Utilization of information and data for different cases.	218
7.1	Evaluating the experiment in Gazebo.	314
7.2	Evaluating the real-world experiment with COBRA quadrotor.	320

List of Figures

3.1	Conditional probability.	39
3.2	State transition diagram or the Bayes net.	42
3.3	Bayes net for a particle filter.	48
3.4	Peaked and highly uncertain distributions.	49
3.5	Bayes net for localization.	56
3.6	Bayes net for mapping.	57
3.7	Bayes net for single-robot SLAM.	60
3.8	View-based SLAM and feature-based SLAM.	62
3.9	Bayes net for a Rao-Blackwellized PF-SLAM.	66
3.10	An example of map merging.	83
4.1	The proposed map segmentation algorithm.	92
4.2	2D Gaussian distribution.	95
4.3	Vertical and horizontal scans for a simple edge map.	96
4.4	CoroBot Robots, equipped with laser rangers and encoders.	108
4.5	A simple test environment.	108
4.6	The first occupancy grid map of the test environment.	109
4.7	The second occupancy grid map of the test environment.	109
4.8	Two different segments form the first map.	110
4.9	Both maps in the same coordinates with marked segments.	110
4.10	Radon image of the first map.	111

4.11	Radon image of the second map.	112
4.12	3D representation of the similarity index.	114
4.13	Final alignment of both maps.	114
4.14	3D representation of the verification index.	115
4.15	Floor plan of the real world test environment.	116
4.16	Three local maps provided by three robots.	116
4.17	Map fusion for the first and second maps.	117
4.18	Map fusion for the first and third maps.	117
4.19	Map fusion of all maps.	118
4.20	Flow chart of the proposed algorithm.	125
4.21	Map segmentation algorithm.	128
4.22	Structure of the SOM.	129
4.23	Surface and point norms.	132
4.24	Map of the first experiment.	138
4.25	Map of the second experiment.	139
4.26	Transformation of the first experiment.	141
4.27	Radon image of the first map.	142
4.28	Radon image of the second map.	143
4.29	Convergence of the Performance index.	144
4.30	A more complex environment with approximate trajectories of the robots.	144
4.31	Maps of the second experiment.	146
4.32	Uncertain transformation.	155
4.33	Linearized transformation uncertainty propagation.	157
4.34	The proposed map fusion algorithm.	159
4.35	GVD and PGVD.	162
4.36	Edges of a GVD.	167
4.37	Experiment 1: Radish dataset.	175

4.38 Experiment 2: two CoroBot robots.	176
4.39 Experiment 3: three CoroBot robots.	177
4.40 Hough modelling.	181
4.41 Property 4.5.1.	183
4.42 The proposed map fusion algorithm.	184
4.43 The overlap of the maps.	185
4.44 Experiment 1: Radish dataset.	195
4.45 Result of circular cross correlation.	196
4.46 Hough images	197
4.47 Experiment 2: simulation in Gazebo.	198
4.48 Experiment 3: two CoroBot robots.	200
4.49 Experiment 4: three CoroBot robots.	201
5.1 Bayes net for SLAM.	215
5.2 Bayes net for multiple-robot SLAM with no line-of-sight observation.	219
5.3 Bayes net for multiple-robot SLAM with line-of-sight observation.	224
5.4 Bayes net for multiple-robot SLAM with line-of-sight observation.	226
5.5 The relations between three robots.	235
5.6 The proposed particle regeneration algorithm.	239
5.7 Sequence of updates and predictions in a particle filter.	250
5.8 The simulated world.	255
5.9 The mean squared error of the position of the landmarks.	256
5.10 Transformation of a feature.	259
5.11 Uncertain transformation of the pose of a robot.	260
5.12 Merging two features.	261
5.13 The hybrid SLAM algorithm.	263
6.1 The evolution of the quadrotor.	268

6.2	The flight of a quadrotor in different directions.	270
6.3	Acting forces on a quadrotor.	271
6.4	Required tasks to accomplish navigation.	275
6.5	Tasks that need to be accomplished towards map learning.	280
7.1	Proposed autonomous navigation.	290
7.2	A sample mission composed of basic navigation behaviours.	292
7.3	Flow chart of rgbdSLAM.	294
7.4	An example of the wavefront algorithm.	302
7.5	An example of an octree structure.	304
7.6	The COBRA quadrotor.	308
7.7	The simulated quadrotor in Gazebo.	309
7.8	Simulation in Gazebo.	310
7.9	Robot position error during mission.	313
7.10	Experiment with a CoroBot robot.	315
7.11	Experiment with COBRA quadrotor.	317
7.12	Paths and trajectory of the robot.	318
7.13	Performance of waypoint following.	319

List of Symbols, Nomenclature or Abbreviations

Notation	Description
$x_{1:t}^a$	the state of robot a from time 1 to time t : $x_{1:t}^a \equiv \{x_1^a, x_2^a, \dots, x_t^a\}$
$z_{1:t}^a$	the observations made by robot a from time 1 to time t : $z_{1:t}^a \equiv \{z_1^a, z_2^a, \dots, z_t^a\}$
$u_{1:t}^a$	the control signals which drive robot a from time 1 to time t : $u_{1:t}^a \equiv \{u_1^a, u_2^a, \dots, u_t^a\}$
$m_{1:t}$	the global map built by all robots on a team from time 1 to time t : $m_{1:t} \equiv \{m_1, m_2, \dots, m_t\}$
$x_t^{a(i)}$	the pose of robot a at time t held by the i^{th} particle
$m_t^{a(i)}$	the local map built by robot a at time t held by the i^{th} particle
$m_{1:t}^a$	the map built by robot a from time 1 to time t : $m_{1:t}^a \equiv \{m_1^a, m_2^a, \dots, m_t^a\}$
$w_t^{(i)}$	the weight of the i^{th} particle at time t
$\mathcal{R}_{\theta=\alpha_1:\alpha_2}(m)$	the Radon transform of the input map, m , over the specified domain of angles, θ
$\mathcal{H}_{\theta=\alpha_1:\alpha_2}(m)$	the Hough transform of the input map, m , over the specified domain of angles, θ
$H(m(i, j))$	the entropy of map m at location (i, j)
$H(m)$	the entropy of map m
$T_{x,y,\psi}(m)$	transformation of a map: the input map m is translated according to x and y along X and Y axes, respectively, and rotated by ψ around Z axis

Abbreviation	Description
ARW	adaptive random walk
EIF	extended information filter
EKF	extended Kalman filter
EVG	extended Voronoi graph
FOG	fiber optic gyros
GVD	generalized Voronoi diagram
GPS	global positioning system
HMM	hidden Markov model
IMU	inertial measurement unit
LIDAR	light detection and ranging
LUP	linearized uncertainty propagation
MAP	maximum a posteriori
MEMS	microelectromechanical systems
MILP	mixed integer linear programming
MM	mathematical morphology
OGM	occupancy grid map
OIF	order invariant function
PDF	probability distribution function
PF	particle filter
PGVD	probabilistic generalized Voronoi diagram
RADISH	robotics data set repository
RFID	radio-frequency identification
RBPF	Rao-Blackwellized particle filtering
RLG	ring laser gyroscopes
ROS	robot operating system
SLAM	simultaneous localization and mapping
SLR	scanning laser ranger
SODAR	sound detection and ranging
SOM	self organizing map
STARMAC	Stanford testbed of autonomous rotorcraft for multi-agent control
UAV	unmanned aerial vehicle
UGV	unmanned ground vehicle
USAR	urban search and rescue
USV	unmanned surface vehicle

Preface

This thesis is based on articles published in international journals and conference proceedings. The chapters presented in this thesis are based on the following publications:

Chapter 3.4 Background on Multiple-robot SLAM

[J1]	Sajad Saeedi , Michael Trentini, and Howard Li. "Multiple-robot Simultaneous Localization and Mapping - A Review." submitted to <i>Journal of Field Robotics</i> .
------	---

Chapter 4.2 Map Segmentation

[J2]	Sajad Saeedi , Liam Paull, Michael Trentini, and Howard Li. "Occupancy Grid Map Merging for Multiple-robot Simultaneous Localization and Mapping." <i>International Journal of Robotics and Automation</i> . 30(2), pp. 1-9. 2015 (in press).
[C1]	Sajad Saeedi , Liam Paull, Michael Trentini, Howard Li. "Multiple Robot Simultaneous Localization and Mapping." <i>IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)</i> . pp. 853-888, 2011.

Chapter 4.3 Neural Networks Approach

[J3]	Sajad Saeedi , Liam Paull, Mike Trentini, Howard Li. "Neural Network-based Multiple Robot Simultaneous Localization and Mapping". <i>IEEE Transactions on Neural Networks</i> . 22(12), pp. 2376-2387, 2012.
[C2]	Sajad Saeedi , Liam Paull, Michael Trentini, Howard Li. "Neural Network-based Multiple Robot Simultaneous Localization and Mapping." <i>IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)</i> . pp. 880-885, 2011.

Chapter 4.4 Probabilistic Generalized Voronoi Diagram

[J4]	Sajad Saeedi , Liam Paull, Michael Trentini, and Howard Li. “A New Efficient Topological Approach to Map Merging Based on a Probabilistic Generalized Voronoi Diagram.” <i>IEEE Robotics and Automation Magazine</i> . 21(2), pp. 60-72. 2014.
[C3]	Sajad Saeedi , Liam Paull, Michael Trentini, Mae Seto, Howard Li. “Efficient Map Merging Using a Probabilistic Generalized Voronoi Diagram.” <i>IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)</i> . pp. 4419-4424, 2012.

Chapter 4.5 Hough Peak Matching

[J5]	Sajad Saeedi , Liam Paull, Michael Trentini, Mae Seto and Howard Li. “Map Merging for Multiple Robots Using Hough Peak Matching.” <i>Robotics and Autonomous Systems</i> . 62(10), pp. 1408-1424. 2014.
[C4]	Sajad Saeedi , Liam Paull, Michael Trentini, Mae Seto, Howard Li. “Map Merging Using Hough Peak Matching.” <i>IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)</i> . pp. 4683-4688, 2012.

Chapter 5 Particle Filter-based Multiple-robot SLAM

[C5]	Sajad Saeedi , Michael Trentini, Howard Li. “Multiple-robot SLAM with Line-of-sight Observations.” to be submitted to <i>IEEE/RSJ International Conference on Robotics and Automation (ICRA)</i> . 2015.
------	---

Chapter 7 Perception, Navigation, and Autonomy

[J6]	Sajad Saeedi , Amr Nagaty, Carl Thibault, Michael Trentini, Howard Li. “Perception and Navigation for Autonomous Rotorcraft in GPS-denied Environments.” <i>International Journal of Robotics and Automation</i> , conditionally accepted.
[C6]	Sajad Saeedi , Amr Nagaty, Carl Thibault, Michael Trentini, Howard Li. “Perception and Navigation for Autonomous Rotorcraft.” <i>International Conference on Intelligent Unmanned Systems (ICIUS)</i> , Montreal, Quebec, Canada, Sep 29 - Oct 1, 2014.

Part I

Introduction

Machine autonomy has been one of the dreams of mankind which has led to outstanding industrial and technological advances. People have sought devices and tools which could do their mundane, repetitive, or impossible tasks such as cleaning, driving, or disposing of nuclear waste. This goal has been pursued through developing robotic techniques. Devices such as driverless cars, mining robots, and extraterrestrial explorers are just a few examples of the realization of this idea. Developing these technologies is not easy for a simple reason: there are many types of uncertainties for robots which need to be accommodated. For example, robot *environments* are very unpredictable, dynamic, and time varying; *algorithmic approximations*, such as linearization, cause incomplete modelling and therefore are another source of uncertainty for robots; robot *sensors* are limited by precision, accuracy, and dimensionality. A camera, for instance, has limited range and resolution. It even distorts some parts of the image due to lens effects. In fact, robotics engineering is all about dealing with these uncertainties.

Most robotic applications require reaching a desired location without human intervention, while taking into account all types of uncertainties. Generally, this is referred to as navigation. In an environment, unknown or partially known to a robot, navigation is more complicated. An autonomous robot needs to address two critical problems to survive and navigate within its surroundings: mapping the environment and finding its relative location within the map. Simultaneous localization and mapping (SLAM) is a process which aims to localize an autonomous mobile robot in a previously unexplored environment while constructing a consistent and incremental map of its environment. Correlation and dependency of localization and mapping on each other raise the complexity of the problem and necessitate accurately solving these two problems at the same time.

As an alternative solution, global positioning system (GPS) provides an earth relative position of the robot; however, there are limitations to its applications. GPS

functions only in outdoor environments with appropriate line of sight to the receiver. This means that it is not practical for indoor, underwater, underground, or extraterrestrial applications. Moreover, the accuracy of the GPS measurements is affected by atmospheric conditions. In contrast, GPS localization has a bounded error. In outdoor applications and especially in large scale environments, it can be combined with SLAM techniques to improve the efficiency and the accuracy of localization and mapping.

While the single-robot SLAM is challenging enough, moving to a platform of multiple robots adds another layer of challenge. In a multiple-robot environment, robots must incorporate all available data to construct a consistent global map, meanwhile localizing themselves within the global map. Multiple-robot SLAM has benefits such as performing missions faster and being robust to failure of any one of the robots; however, these benefits come at the price of having a complex system which requires coordination and cooperation of the robots. In this work, a few challenges in multiple-robot SLAM are investigated, and solutions for these challenges are proposed.

Most works on SLAM have focused on ground robots, where in most cases the motion of the robot is limited to the 2D plane. This becomes more complicated when SLAM is performed by a flying robot, such as a quadrotor rotorcraft. The added complexity derives from the inherent limitations of such robots. Limited payload, 3D motion, and lack of direct odometry are just a few examples of such limitations. Moreover, the quadrotor SLAM can be even more complicated if the robot is supposed to perform autonomous navigation on top of SLAM. As another part of this research, these challenges are investigated and an integrated solution for perception and navigation by an autonomous quadrotor is proposed.

This thesis studies the possibility of using state-of-the-art sensing technologies to perform SLAM in GPS-denied environments for two different scenarios: 1) a team of ground robots, and 2) an autonomous quadrotor rotorcraft. The thesis has been

organized in four parts. This Part presents the introduction. The introductory work is organized in two chapters: Chapter 1 presents the introduction to the multiple-robot SLAM project, and Chapter 2 is the introduction to perception and navigation of an autonomous quadrotor. Part II presents the multiple-robot SLAM. Part III demonstrates the autonomy solution for the quadrotor. Finally, Part IV presents conclusions and the future work for both area of investigation.

Chapter 1

Multiple-robot SLAM

This chapter presents an introduction to multiple-robot simultaneous localization and mapping. In Section 1.1, the problem statement is presented. Section 1.2 presents the significance and objectives of the research. Section 1.3 reviews the literature. Section 1.4 introduces the proposed methods. Finally, Section 1.5 summarizes the contributions of the work.

1.1 Problem Statement

So far different solutions have been proposed for SLAM [1]. However, most of these solutions consider single-robot SLAM; few of them have considered multiple-robot SLAM. Multiple-robot SLAM is motivated by the fact that exploration and mapping tasks can be done faster and more accurately by multiple robots than by a single robot. In addition, in a distributed system, the whole team is more robust since the failure of one of the robots cannot halt the entire mission [2]. Collaboration-based operations like fire fighting in forested and urban areas; rescue operations in natural disasters; cleaning operations like removing marine oil spills; underwater and space exploration; and security, surveillance, and maintenance investigations need to be completed fast and autonomously and require localization and mapping.

While single-robot SLAM is challenging enough, moving to a multiple-robot SLAM platform adds another layer of challenge. In a multiple-robot environment, every robot should achieve its own specific goals to provide an accurate estimation of its surroundings and current position while contributing to the global mission. This requires each robot to react to information about other robots. Furthermore, each robot must incorporate new information into its decision structure to react to environmental changes.

Potential problems in multiple-robot SLAM are listed and explained briefly. Some of these problems are the focal points of the research, and the proposed solutions try to handle these issues to the extent possible. The next subsections introduce these problems.

1.1.1 Relative Poses of Robots

In multiple-robot SLAM, the map provided by each robot in its own reference coordinates is called the local map. Each local map is generated from coordinated measurements such as laser scans. Each robot tries to integrate all of the local maps provided by the other robots to generate a global map of the environment. However, this is a difficult task because the required alignments or transformation matrices which relate these maps to each other are in general unknown.

1.1.2 Uncertainty of the Relative Poses

Uncertainty, which according to Thrun *et al.*, “arises if the robot lacks critical information for carrying out its task” [3], is identified as having five main roots: environment, sensors, robots, models, and computations. There is an uncertainty associated with the relative transformation matrix represented in the form of a covariance matrix. This uncertainty is mainly because of modelling uncertainties, sensory noises and linearization effects. Updating the maps and poses should be performed using the

covariance matrix. Depending on the method used for finding the relative transformation matrix, covariance matrix takes different values.

1.1.3 Updating Maps and Poses

Once the relative transformation is found, a procedure is required to fuse local maps. The resulting map should integrate all information from given local maps. As a result of updating the maps, poses of the robots should also be updated. This requires considering the trajectory of the robots and new information received from other maps. Due to the nature of multiple-robot SLAM, updating poses and maps is a coupled problem.

1.1.4 Line-of-sight Observations

On some occasions, robots can see and detect each other. Each robot might already have an estimate of the pose of other robots. However, when robots can see each other through line-of-sight and direct observations, these estimates can be improved. This fact can help robots to reduce mapping and localization error.

1.1.5 Closing Loops

Loop closure, also called cycle detection, is defined as identifying a place observed previously but not very recently (recently is defined in relation to the mission duration). Loop closure for a single robot is challenging enough. Extending this problem for a team of multiple robots requires solving it using all resources of information from individual robots.

1.1.6 Complexity

Robotics applications are usually real-time. Thus, it is very important to design an algorithm capable of solving the above mentioned problems with minimum time and memory requirements. In multiple-robot SLAM, space complexity and time complexity are two important issues. The complexity of a multi robot algorithm directly affects its scalability.

1.1.7 Communications

Availability of a medium for data sharing among robots is an important requirement in multiple-robot SLAM. Information between robots can be exchanged via communication channels. The quality of the communication channels is dependent on the environment. For instance, communication issues are a significantly challenging problem for a team of robots in underwater environments, where the environment imposes limitations on the bandwidth and data rate.

1.2 Significance and Objectives

Modern robots have different applications in industrial, military, and domestic settings. For mobile robots like cleaners, servants, and mine detectors, which are often deployed in large numbers, having reliable perception is key to achieving required tasks. Therefore, multiple-robot SLAM as a solution to perception issues can have high impact on the applications of robots.

Meanwhile using swarms of autonomous robots is an inexpensive alternative to having humans perform risky and hazardous tasks. Often we hear news about miners trapped in mines or workers losing their lives in petroleum refineries or power plants. Extraterrestrial applications like space exploration are also highly risky for humans. By deploying robots to conduct dangerous tasks, risks can be minimized.

Unless for recharging or refuelling, autonomous robots do not need to sleep or rest; this can help countries to develop faster and more sustainably. Since human labour is becoming more expensive, countries putting more effort into developing autonomous robots will get greater benefit in the long term.

The overall motivation of the research is to expand the level of robot autonomy. The goal of the proposed work is to develop a multiple-robot SLAM algorithm based on single-robot SLAM to explore an unknown environment autonomously. The proposed work will be an autonomous multiple-robot system to perform integrated SLAM.

1.3 Literature Review

Robotic mapping started with mapping features (landmarks) in the environment. Features can be points or any kind of geometrical shapes extracted from measurements. A map constructed by features is often called a feature map. Since it is important to show uncertainty in the maps, occupancy grid mapping, introduced in 1987 [4], integrates uncertainty of the measurements into the maps. Occupancy grid maps are very effective when robots are required to explore and map unstructured environments where it is difficult to extract features. Also grid mapping handles dynamic and moving parts of the environment efficiently [5]. By introducing topological maps, it became possible to combine different maps into hybrid maps which facilitate map analysis and path planning for robots [6].

The evolutionary history of robotic localization involves three main steps: dead reckoning, *a priori* localization, and SLAM. Dead reckoning is the estimation of the vehicle pose by integrating estimates of its motion using encoders or other sensors. The result of dead reckoning is not accurate due to the error accumulation in the integration process. This problem is really serious in the case of large outdoor environments. With the advent of GPS technology, localization based on the *a priori* maps became

popular, but still the accuracy was low and there was no solution for indoor applications. Also, this method requires exploring the environment in advance, which is not possible in some applications like space exploration. The need to overcome the limitations of *a priori* mapping made researchers consider mapping and localization one coupled problem. In 1987, for the first time, localization and mapping were formulated with the presence of uncertainty and erroneous measurements [7].

SLAM techniques are either feature-based or view-based. In feature-based SLAM, features from observations are extracted and used for localization while in view-based SLAM, observations are processed without extracting any features. Each has its own advantages. In feature-based SLAM, special features such as points, lines, or corners are extracted from the surroundings. Within consecutive observations, these features are used as landmarks to compensate for the localization error. The advantage of this method is its dependency on landmarks which have fixed position and hence results in better localization. This method requires feature extraction and data association of different features, which make it complicated and create a high computational demand. Meanwhile, feature extraction algorithms have to change in different environments [8–12]. In view-based SLAM, which is usually used with dense range-bearing measurements, such as laser scans, no feature is extracted from the data directly. This method is motivated by the fact that in some environments, it may be difficult to define appropriate parametric feature models. In view-based SLAM, using image processing techniques for data association, raw data is processed to provide mapping and localization information. Since no feature is extracted, no information is lost and there is no need to extract geometrical features. But processing raw data requires high computational power which results in low-speed navigation while doing on-the-fly view-based SLAM [13–17]. Another form of the view-based SLAM is called appearance-based SLAM, which is often performed with optical sensors. In appearance-based SLAM the appearance of the current view is compared with the

available views (reference images) to localize the robot [18, 19].

Multiple-robot SLAM is one of the interesting developing areas in robotics which has attracted researchers recently. The majority of the multiple-robot SLAM methods utilize feature-based SLAM. A feature-based multiple-robot SLAM algorithm is proposed which is based on the information filter [20]. Zhou *et al.* propose a classic EKF-based algorithm for feature-based multiple-robot SLAM [21]. Gil *et al.* present a feature-based visual SLAM using particle filtering [22]. A feature-based multiple-robot SLAM is proposed in [23] where fuzzy sets are used to represent uncertain position information and fuzzy intersection is used to fuse data. In [24] an algorithm for merging feature-based maps with limited communication is introduced.

Map merging or map fusion [2] is a solution to multiple-robot SLAM in which map data from robots are fused to generate a global map. To fuse maps, the transformation matrix between maps must be found. To find the required transformation, the first step is to identify overlaps and then use the overlaps to calculate the alignment. The map merging problem can be interpreted as a special case of image registration in computer vision where images are maps with special geometry. However, the problem is that in multiple-robot map merging, the percentage of overlap between maps is usually low. Alternatively, map merging can be thought of as a solution to find the transformation between maps.

The rest of the literature review focuses on the following key problems in view-based multiple-robot SLAM.

1.3.1 Relative Poses of Robots

Birk *et al.* present a solution based on occupancy map merging [2]. This method uses map-distance as a similarity index and tries to find similar patterns in two maps based on a simple random walk algorithm. The main drawback of this method is that it usually fails especially when there are few similar patterns in both maps.

The method proposed in [25] is based on the solution proposed in [2], except that individual robots use distributed particle SLAM (DP-SLAM) version 2.0 [26] for filtering. In [27], multiple-robot SLAM based on topological map merging using both structural and geometrical characteristics of the Voronoi graph is proposed. In this case the topological map is built on the occupied space as opposed to the free space. The assumption in this work is that a robot will be able to recognize areas of the map that correspond to vertices.

1.3.2 Uncertainty of the Relative Poses

To the best of our knowledge, there is no known research in multiple-robot SLAM that specifically formulates and takes into account the line-of-sight observations between the robots.

1.3.3 Updating Maps and Poses

In [28], a method is proposed using a particle filter. In this method, it is assumed that robots will meet each other at a point. At the meeting point robots know their relative positions and from this point a particle filter is applied to the data in reverse temporal sequence to find their relative initial poses. The proposed method finds the initial state of the robots but it cannot be applied when robots cannot see each other.

In [29], a method similar to that of [28] is used but with a central agent and 80 robots. In [25] only maps are updated by applying a simple rule: the minimum occupancy grid value between the two maps is used for merging. For example an occupied cell in one map is assumed to be occupied in both maps.

1.3.4 Line-of-sight Observations

To the best of our knowledge, there is no known research in multiple-robot SLAM handling this problem specifically.

1.3.5 Closing Loops

Loop closure in different approaches is treated differently. In [28] and [29], loop closure is performed by having multiple hypotheses for data association through particles. In [2, 25, 30, 31], the loop closure is performed indirectly through merging maps. This approach is not flexible enough to accommodate errors accumulating in maps; however, it can generate accurate global maps if the single robot SLAM, performed on each robot, is reliable.

1.3.6 Complexity

The method proposed in [2] and used in [25] is highly time consuming. This is problematic especially for the large scale maps which are a typical problem in indoor environments. The method proposed in [28] is moderately fast but demands high computational power and memory since it is based on particle filtering. The method in [27] is claimed to be fast. This is mainly because the map and pose are not updated and no uncertainty for the transformation matrix is assumed.

1.3.7 Communications

Communications limitations are an important issue in multiple-robot SLAM. Specifically in environments such as underwater, low-bandwidth and low-speed communications impose limitations on accuracy and scalability of team-based operations. Leung *et al.* studied communications problems in multiple-robot localization [32] and multiple-robot SLAM [33]. In their approach, a centralized-equivalent estimate

is obtained by all robots in a decentralized manner assuming the network is never fully connected. This solution enforces delay on the state estimation; however, by applying the Markov property at certain times, a robot does not need to keep track of what other robots know. In [24] an algorithm for merging feature-based maps with limited communication is introduced. Other aspects of communications issues have been studied by Bar-Shalom [34], Bar-Shalom *et al.* [35], and Howard *et al.* [36].

1.4 Proposed Method

In this research, for some of the identified problems, solutions have been proposed. The proposed solutions are classified based on the identified problems.

1.4.1 Relative Poses of Robots

For this problem four solutions are proposed. The key idea in these methods is to find the relative poses by comparing the maps of the robots.

- Map segmentation: this is an algorithm which is performed by identifying edges in the maps and introducing a smoothing method to extract unique characteristics of occupancy grid maps. These unique characteristics are used to find the relative transformation matrix. The motivation for this approach is to extract identical segments of the maps and then use those segments to find the relative poses.
- Self-organizing map (SOM) SLAM: this method is a neural network-based map clustering approach to scale down and extract features of occupancy grid maps for map fusion and multiple-robot SLAM. This method is based on competitive self-organizing maps. There is no known similar research or application. The logic behind this approach is that by learning the maps, using neural networks,

the maps are scaled down. The positions of the neurons, which represent a model of the map, can then be used to find the relative poses.

- Probabilistic generalized Voronoi diagram (PGVD): PGVD is a novel probabilistic map fusion algorithm based on the generalized Voronoi diagram (GVD) and is applicable to 2D and 3D environments. In this method, morphological operations are used to build PGVD for occupancy grid maps. Then PGVDs are cross-matched to find the transformation matrix. This algorithm was motivated by two facts: first, a topological map represents the salient information of the maps; therefore, a topological map is processed faster than other types of the maps, second, by extending the topological mapping to a probabilistic framework, the uncertainty of the mapping is propagated to the topological maps. Thus, a probabilistic topological map allows us to take into account the uncertainty of the mapping process.
- Hough peak matching: In this approach maps are transformed into the Hough space. Using the properties of the transformation in the Hough space and by matching local peaks of the Hough images, the required alignments are calculated. Experimental results show that this is a fast and robust approach. The rationale for the Hough peak matching rests on the fact that the Hough transform models the lines in the occupancy grid maps as peak points. In other words, the configuration of the Hough peak points represents a model of the world. In general, dealing with points is easier than lines; therefore, by processing the peak points, overlaps of the maps are identified efficiently.

1.4.2 Uncertainty of the Relative Poses

For this problem, linearized uncertainty propagation (LUP) is proposed. Given the transformation matrix and its uncertainty as a covariance matrix, a formulation is

proposed to consider the effect of the uncertain transformation on the transformed map. LUP is motivated by the fact that the nonlinear transformation can be linearized using the Taylor expansion. Then the uncertainty of the relative transformation is propagated to the transformed map using the linearized transformation function.

1.4.3 Updating Maps and Poses

For this problem, an entropy filter is proposed. After finding the transformation matrix, an entropy filter is used to update occupancy grid maps. The entropy filter integrates the information from multiple sources, considering the change in the entropy. Also a novel scalable solution using a particle filter is proposed. This solution improves the particle filter to accommodate a team of robots performing SLAM. The proposed solution can be used either for feature-based or for view-based SLAM. Particle filter has the advantage that it can model multimodal distributions, and also it represents nonlinear systems efficiently.

1.4.4 Line-of-sight Observations

When robots are in line-of-sight of each other, a new source of information becomes available. A method is proposed which processes line-of-sight observations to improve mapping and localization. By taking into account the line-of-sight observations, the trajectories of the robots become coupled, and the accuracy of the poses increases.

1.4.5 Closing Loops

For loop closure two solutions are proposed. The solution suite proposed for finding the relative transformation solves loop closure indirectly. Also, a novel particle filter is proposed which can be applied to a team of robots and performs loop closure cooperatively and efficiently.

1.4.6 Complexity

Each of the proposed methods requires processing and memory resources. These resources are compared to analyze the complexity of the algorithms. Also the proposed particle filtering improves upon traditional methods by employing fewer particles and hence improving the complexity and scalability of the proposed solution.

1.4.7 Communications

It is shown that in certain cases, a particle filter can deal with out-of-sequence measurements more efficiently. Also, cyclic updates, which result in overconfidence, are avoided by minimizing sharing the processed information between the robots.

1.5 Contributions

The major contributions of the proposed work are introduced in this section. These contributions address issues stated in the problem statement and are explained in the following chapters in detail.

- Map segmentation: This new approach finds the relative transformation in two steps. First an approximate transformation is found. Then it is tuned. To find the approximate transformation, edges of maps are extracted. Using a novel smoothing algorithm, edges are smoothed to reduce computational demand. The smoothed edges are filtered through a novel histogram filter. The histogram filter chooses segments with features. The segments are matched against each other to find an approximate transformation. Then the transformation is tuned utilizing the Radon transform. At the end, the results are verified.
- Self-organizing map SLAM: This new approach reduces the dimensionality of the maps by clustering them. Then a novel approach is proposed to match

clusters and find the relative transformation. In the literature, there is no similar method which uses neural networks for map fusion.

- Probabilistic generalized Voronoi diagram: This innovative algorithm is designed to consider the probabilistic nature of the maps and preferentially match parts of the maps which have higher certainty. Using a novel approach, the GVDs of maps are extended to reflect their probabilistic properties. Then edges of the probabilistic GVDs are matched to find the relative transformation. A new verification method is proposed to verify the matching results.
- Hough peak matching: This method is a new approach that finds the transformation in the Hough space rather than Euclidean space. First the maps are transformed into the Hough space. Then, in the Hough space, features of maps are selected. A novel feature matching algorithm is proposed which matches features of the maps in the Hough space. This solution is fast and robust and can work in different environments. There is no similar method of matching occupancy grid maps.
- Linearized uncertainty propagation: When a map is transformed with an uncertain transformation matrix, the resulting transformed map should reflect the uncertainty of the transformation. LUP is a new approach which applies uncertainty of the transformation for all cells of the transformed map. This new approach linearizes the transformation and propagates the uncertainty to the transformed cells.
- Fusing aligned maps: Through a new formulation and utilizing the additivity property of *log odds* of the occupancy grid maps, an entropy filter is proposed which efficiently fuses two aligned maps.
- Particle filter-based multiple-robot SLAM: This method is a novel approach

which efficiently solves two important problems: line-of-sight observations and loop closure. In this method, a novel structure for particle filtering is proposed which takes into account line-of-sight observations between robots and performs loop closure in a multiple-robot environment. This novel approach can be utilized both for view-based and feature-based SLAM.

Chapter 2

Rotorcraft SLAM

This chapter is an introduction to quadrotor rotorcraft perception and navigation. In Section 2.1, the problem statement is presented. Section 2.2 presents significance and objectives of the research. Section 2.3 reviews the literature. Section 2.4 outlines proposed methods. Finally, Section 2.5 summarizes the contributions of the work.

2.1 Problem Statement

Autonomous navigation and perception are studied for a special type of rotorcraft called the quadrotor. The quadrotor has four rotors, by which the robot is stabilized and controlled. To perform SLAM and autonomous navigation with a quadrotor, one needs to address a set of problems and limitations inherent to the quadrotor, which are discussed in the following sections.

2.1.1 Nonlinear and Fast Dynamics and Vibration Effects

In addition to the cross couplings of the gyroscopic moments, quadrotors are also highly nonlinear. Also it is an underactuated and a fast system; thus, to have a reliable flight, a proper controller is required. Moreover, vibration due to the fast

spinning rotors affects measurements, especially the inertial sensors.

The frequency of the vibration is directly related to the rotation rate of the propellers. The controller constantly changes the rotation rate of the propellers to stabilize the quadrotor; therefore, the frequency of the vibration is also changing. The amplitude of the vibration is directly affected by the quality of the propellers and the structure of the rotorcraft. Due to variations during manufacturing, a propeller is never perfectly symmetrical and its individual blades are not perfectly balanced in weight and strength. The greater the deviation is, the more intense the vibration becomes. Additionally, a rigid propeller deforms less under load; thus, it produces less vibration than a soft propeller. A proper vibration damping system can significantly reduce the transmission of the vibration to the sensor suite.

2.1.2 Limited Payload

Onboard sensors such as the laser ranger, camera, and inertial sensors together with the power source and processing units are heavy and require considerable space. To overcome this limitation, small and lightweight sensors and processing units must be mounted on the robot which result in low quality sensing and slow processing. Batteries are the main power source of a flying robot. Because of the limited payload, small and lightweight batteries must be carried onboard; therefore, the time of the flight is very limited.

2.1.3 Odometry Limitation

Ground robots are usually equipped with odometric sensors which provide an estimate of the pose of the robot. Pose estimates based on odometry measurements are in general unreliable due to slippage and non-smooth terrain; however, these estimates can be filtered using filtering techniques combined with the motion model of the robot. Moreover, having an approximate estimate of the motion of the robot is in general

better than having no estimate. Flying robots cannot have such a direct odometry measurements. Alternatively, visual or laser odometry and inertial measurements are used to overcome this problem.

2.1.4 Perception

As mentioned, for autonomous flying robots, there exist many challenges such as fast and nonlinear dynamics that require a reliable controller; limited sensing payload, which affects the quality of measurements; and vibration effects, which make measurements noisy. Another key challenge for autonomous navigation is the need for knowledge of the position of the robot. In GPS-denied environments this is achieved by simultaneous localization and mapping where the robot develops a map of the world and situates itself in it. Localization and mapping should happen in real time and their accuracy will affect the performance of the navigation controller. Therefore, reliable real-time localization and mapping in GPS-denied environments is a cornerstone of autonomous flying robots.

2.1.5 Navigation

For a quadrotor, autonomous navigation is a complicated requirement. To provide this requirement, a set of problems must be addressed. These problems include mission planning, map learning (*i.e.*, path planning, localization, and mapping), state estimation, and control. These problems depend on each other. For instance, the control needs the current position and orientation of the robot which is provided by the localization, mapping, and state estimation. Providing an efficient solution for each of these problems must take into account the limitations of the quadrotor.

2.2 Significance and Objectives

Flying robots will revolutionize the robotics world in the near future. This is mainly for two important reasons. First, unlike ground robots which are limited by obstacles and cluttered environments that quickly become untraversable, flying robots can easily overcome these problems. They can pass through windows, fly over mud and water, and manoeuvre around obstacles. Second, the perception of ground robots is limited to scenes and features close to ground level, while flying robots can change their operating attitude and develop perception from all possible vantage points.

In general, quadrotors have limitations, such as limited flight time and payload capacity; however, they have many different applications. As a fast first-responder, it is used by security and police forces to record scenes; as a manoeuvrable monitoring robot, it is used by oil and energy industries to monitor the health of their facilities. If the quadrotor can perform these tasks autonomously and without relying on GPS, it will have even more applications.

The objective of this work is to develop an autonomy solution for a flying quadrotor in such a way that it can perform autonomous navigation in an unknown and GPS-denied environment. The sensor suite and the autonomy solution must consider the limitations of the quadrotor.

2.3 Literature Review

There has been an extensive amount of research on 2D perception and navigation specifically for ground robots [3]. However, this is challenging for small aerial vehicles, where there exist payload and computational limitations, there is no direct odometry and the motion is not limited to 2D space. Within the past decades, as a result of small and fast computers being combined with microelectromechanical systems (MEMS) sensors, researchers have developed a great interest in perception and navigation for

flying robots. These robots are often equipped with different sensors such as inertial measurement unit (IMU), scanning laser rangefinder, stereo-camera, altimeter, and compass. The reviewed papers follow the right-handed coordinates system unless otherwise stated. In this system, rotations around X , Y , and Z axes are referred to as roll, pitch, and yaw angles, respectively.

2.3.1 Localization and Mapping

Thrun *et al.* [37], pioneered laser mapping by a low altitude flying helicopter. In their work a GPS, an IMU, a scanning laser rangefinder and a compass were used to perform laser mapping. The scan alignment is performed in a probabilistic framework. Unlike most other works in which scans are aligned in the $X - Y$ plane, in their work scan alignment was done in the $X - Z$ plane and in another step yaw and pitch angles were recovered.

In the Mikrokopter project [38], Grzonka *et al.* developed Monte-Carlo localization and graphSLAM for an indoor flying quadrotor. Their sensing platform includes a scanning laser rangefinder and an IMU. Generally, the IMU generates accurate roll and pitch angles, so in their work scans were projected onto a 2D plane using roll and pitch angles. Then by scan matching, x , y and yaw angles were estimated. Altitude estimates were provided by reflecting a few laser beams to the ground and then filtering with a Kalman filter (the Kalman filter is explained in Chapter 3). Their work is also open-source. The work later was extended in [39] to be fully autonomous. Bachrach *et al.* [40] developed a navigation system for a small quadrotor using a scanning laser rangefinder, a custom built stereo-camera rig, a colour camera, and an IMU. Laser odometry is performed based on the work by Olson *et al.* [41] and visual odometry is done using features from accelerated segment test (FAST) [42]. Using an extended Kalman filter, odometry information is fused with the IMU. The filtered pose is used to provide localization and mapping using Gmapping [43]. The frontier

exploration algorithm [44] is used to explore the unknown environment. The path to achieve waypoints is designed by dynamic programming in the information space [45]. Dryanovski *et al.* [46] developed a pose estimation system for a quadrotor micro air vehicle. In their method, three sensors are used: an IMU, a scanning laser rangefinder and a pressure altimeter. Given roll and pitch angles from the IMU, scans from the laser ranger are projected onto a 2D plane and a fast scan matching is used to perform 2D laser odometry [47]. A few laser beams are projected by a mirror to the ground to measure the altitude. The altitude measurement from the laser and the pressure altimeter are filtered by a Kalman filter. Gmapping [43] is also used to produce a 2D occupancy grid map. This work was developed under robot operating system (ROS) and is open-source.

In the Hector package [48], a fast solution for full 3D pose estimation was proposed. In this work, a scanning laser rangefinder and an IMU are the main sensors. The Hector package is flexible and was developed such that it can use GPS, compass, altimeter, and other sensors. The package was developed under ROS and is open-source. Their method has been tested in many different scenarios including unmanned ground vehicle (UGV) in rough terrains, unmanned surface vehicle (USV) in littoral waters, and a handheld embedded mapping system in indoor environments.

The autonomous urban search and rescue (USAR) platform [49] uses an Ascending Technologies Pelican quadrotor, equipped with a scanning laser ranger, an IMU and stereo cameras. The platform uses a canonical scan matcher by Censi [47] for the laser odometry and a correlation-based algorithm is used for the visual odometry. The results are fused by a Kalman filter. Three layers are defined for autonomous navigation: perception, which performs odometry and data fusion; cognition, which is responsible for path planning and controlling the mission; and action, which takes care of the controller. One experiment in a combined indoor and outdoor environment is presented to evaluate the proposed algorithm.

2.3.2 Path Planning

An important part of any autonomy package for a quadrotor is the motion or path planning. In most applications, the quadrotor is flown manually and the path is generated by a pilot. There exist a few papers developing a path planning algorithm for autonomous quadrotors. Generally path planning algorithms are categorized into two main groups: reactive and deliberate. Reactive methods such as bug algorithms, wavefront, and potential fields are fast and easy to implement while deliberate methods such as cell decomposition, A^{*} search, and genetic algorithms are complicated and computationally expensive. The preference of the path planning method depends on the available onboard processing power and the level of the autonomy. Due to the ability of quadrotors to hover in one location and move in any direction with ease, path planning algorithms are generally designed in the $X - Y$ plane.

Bachrach *et al.* [40] have applied dynamic programming in the information space [45] to generate path. In [50], A^{*} search on visibility graphs and fast marching potential field were implemented. Grzonka *et al.* [39] applied D^{*} lite [51] which is a variant of the A^{*} algorithm which utilizes previous results to improve an invalid plan. Vitus *et al.* [52] developed tunnel-mixed integer linear programming (MILP) path planning in obstacle-rich environments to increase autonomy of the Stanford testbed of autonomous rotorcraft for multi-agent control (STARMAC). In tunnel-MILP, first a path without any consideration of vehicle dynamics is generated. Then, a tunnel through which the robot travels is built as a sequence of convex polytopes. Finally, MILP is utilized to generate a dynamic trajectory which keeps the path of the robot inside the tunnel.

Whether the flight is autonomous or not, controlling the rotorcraft is an important task to achieve a successful flight. This task is out of the scope of this work and readers can find more information in [53–55].

2.4 Proposed Method

The proposed solution addresses all essential requirements to perform perception and navigation by a quadrotor. All measurements from the sensor suite are processed to help the robot perceive its environment.

Dealing with nonlinear and fast dynamics is mostly a controls problem; however, to have an effective control algorithm, it is crucial to know the current position of the robot. Therefore, dynamics problems are tightly coupled with perception problems. The more accurate and frequent is the perception feedback, the more efficient will be the control algorithm. In this work, pose feedbacks for the controller are provided at an average rate of 40 Hz which experimentally is shown to be enough for the controller.

Vibration effects result in noisy measurements, especially from the inertial measurement unit. Although the vibration is reduced by choosing a proper damping system, to minimize its effects a noise removal filtering procedure is required. To filter measurements, a Kalman filter is used which fuses inertial measurements with laser and optical observations. The Kalman filter takes into account the 3D motion of the robot and therefore accounts for physical limitations imposed by the dynamics of the system.

To deal with payload limitations, a lightweight and therefore a low-speed computer is carried onboard. This means demanding processes such as mapping and localization cannot be performed onboard. Instead, the data is captured by the onboard computer and transmitted to a ground station where they are processed and the results are sent back to the onboard computer. The ground station, which is composed of a computer and a wireless router, must be able to communicate with the quadrotor constantly. The stabilization and control algorithm, which needs to run at a high frequency, is performed onboard. The sensor suite has been chosen to be lightweight so that the rotorcraft can carry it. More information about sensors will be provided later.

Lack of direct odometry is compensated for by the laser and visual odometries. The laser odometry is provided by matching consecutive laser scans. To do this, inertial measurements are filtered and used to project the laser scans onto a 2D horizontal plane. Then scans are matched in the 2D horizontal plane, providing change in heading and 2D horizontal translation values. The visual odometry is produced by processing features from consecutive image frames from the onboard camera. This provides full change in orientation and translation values.

To provide reliable perception, two types of mapping and localization are used. One is view-based SLAM, in which laser odometry is used to generate an occupancy grid map and then the robot is localized within the map. The other is feature-based SLAM, where visual odometry is used to produce a map of features and the robot is localized with respect to the features. The results of both SLAM algorithms are fused by a Kalman filter. Relying on two SLAM algorithms requires high computational demand and memory, but has the advantage that if one of SLAM algorithms fails, the other one can still provide perception for the robot.

Based on the perception algorithms, the robot makes a model of the unknown world and uses the model to localize itself within the world. Then a path planning algorithm is customized and used to guide the robot from one point to another. The path planning algorithm, which runs on the ground station, has to be efficient enough to run on-the-fly. Moreover, it should be able to provide a path far enough from obstacles and avoid local minima. To provide all these requirements, an algorithm based on wavefront path planning is proposed and developed. The algorithm uses the occupancy grid map, provided by the view-based SLAM, to perform path planning while the quadrotor is flying. The planned path has a safe distance from all obstacles. Having localization, mapping, and path planning integrated enables the robot to do more complicated missions, composed of a finite set of basic behaviours such as exploring an unknown environment, moving towards special target points, and

returning to the start point. By sequencing these behaviours, many complicated missions can be designed. For instance, a robot can explore its environment, locate a special target, and finally return to the start point to report the mission.

As a basic behaviour, the exploration algorithm is based on the frontier algorithm where the borders between known and unknown parts of the map are used as the next waypoints. These borders are clustered, and the center of each cluster is considered as a potential waypoint. Out of these waypoints, a waypoint which is closer to the robot and its associated border is bigger than others is chosen as the next exploration waypoint.

The proposed method constructs an autonomy solution for the quadrotor which can also be used for ground robots. Extensive experiments, including realistic simulations and real-world experiments on a ground robot and a quadrotor, demonstrate the effectiveness of the proposed perception and navigation solution.

2.5 Contributions

This research proposes an autonomy solution for a flying robot in GPS-denied environments.

- At the mission level, a set of behaviours are developed and structured to suit the goals of autonomous navigation.
- Main building blocks of the autonomy solution are implemented to make the mission goals operational.
- Localization and mapping are performed using mounted sensors considering 3D motion of the robot.
- To guide the robot between waypoints a path planning algorithm is developed.

- The robot has been enabled to explore and map unknown environments using the path planning, localization, and mapping capabilities.
- The proposed work is designed in such a way that it can be utilized on aerial and ground robots.

Part II

Multiple-robot SLAM

Simultaneous localization and mapping is the cornerstone of robotic applications in GPS-denied environments. It enables mobile robots to perceive the environment and perform autonomous tasks without a global positioning reference. However, SLAM is not an easy problem; the dependency of localization and mapping on each other evokes the chicken-and-egg problem, where to have the map, localization is needed and to perform localization, the map must be available. Moreover, extending localization and mapping to a team of robots, adds another layer of challenge to the existing problems. In multiple-robot SLAM, data from all robots and resources have to be gathered and integrated to perform collaborative localization and mapping.

Despite all these challenges, multiple-robot SLAM has very important benefits and applications. It enables the robots to know about and interact with each other. This capability helps the team to achieve their goals more efficiently; they can do the tasks faster, more accurately, and in a robust way.

This part of the thesis proposes novel methods for simultaneous localization and mapping by a team of robots. The proposed work is presented in three chapters: Chapter 3 presents the background information to simultaneous localization and mapping, Chapter 4 introduces four proposed solutions for map merging, and finally Chapter 5 describes the particle filter-based multiple-robot SLAM algorithm.

Chapter 3

Background

This chapter presents background to multiple-robot simultaneous localization and mapping. In Section 3.1, some information about sensing devices is presented. Section 3.2 presents background to the probabilistic estimation. Background to single-robot SLAM is presented in Section 3.3. In Section 3.4 background to multiple-robot SLAM is provided. Finally, Section 3.5 summarizes the chapter.

3.1 Sensing Devices

Sensors are widely used in our daily life to assist us in interacting with different tools and devices. A sensor is a device that measures a physical quantity such as the distance from an object and converts it into an analog or a digital signal. Sensors for a robot are the main tools to acquire information from the environment. In the following subsections, commonly used sensors in robotic applications are introduced briefly.

3.1.1 Scanning Laser Ranger

A laser ranger or a laser rangefinder is a laser device that uses laser beams to find the distance to an object. Generally, laser rangers use phase shift of multiple frequencies reflected from obstacles to measure a distance. A scanning laser ranger (SLR) sensor, also called a light detection and ranging (LIDAR) sensor, is composed of a laser rangefinder and a rotating mirror, which guides the laser beam. The rotating mirror reflects the laser beam such that a 2D plane is scanned, allowing the sensor to cover its surroundings. SLR sensors are very accurate and expensive.

In robotics applications, Hokuyo UTM-30LX [56] is a commonly used scanning laser ranger. It operates at 40 Hz and covers 270°. It can detect up to 60 m, but only a maximum range of 30 m is guaranteed. Its range accuracy is ± 50 mm, and its angular resolution is 0.25°. It weighs about 370 g.

3.1.2 Sound Detection and Ranging

A sound detection and ranging (SODAR) sensor is a device which uses sound propagation techniques to determine the distance to an obstacle. SODAR sensors use air or water to propagate sound signals. Usually SODAR sensors use time-of-flight to calculate the distance. Compared to laser rangers, SODAR sensors are inexpensive but less accurate.

As an example, the SensComps MINI-A PB Ultrasonic Transducer [57] is a SODAR sensor that can detect obstacles within the range of 0.3 m to 3.5 m. This sensor is a 50 kHz electrostatic transducer.

3.1.3 Optical Sensors

Optical sensors have various features and applications. Colourimeter, infra-red detector, phototube, and scintillometer are a few examples of optical sensors which

operate at different wavelengths of light. In robotics, monocular and stereo cameras are widely used to perceive the environment. A stereo camera has the advantage that it can provide range and bearing information for each pixel, while a monocular camera provides only the bearing information. They are relatively inexpensive; however, further processing is required to analyze images. Both types of sensors are subject to lens effects such as image distortion.

Infrared or thermographic cameras are another type of optical sensors which generate images using infrared radiation. Infrared cameras are mainly used in military applications such as night vision and target acquisition.

3.1.4 Magnetometer

A magnetometer is a sensing device that measures the strength and/or direction of magnetic fields. Magnetometers are typically used to measure magnetic fields of the Earth. In robotic applications, magnetometers are used to determine the orientation of a robot with respect to the Earth's magnetic fields. Magnetometers are widely used in underwater applications; however, in indoor robotic applications, they suffer from poor performance due to existence of magnetic metals.

Early magnetometers had mechanical structures and were relatively large. In recent years, magnetometers have been miniaturized and integrated with electronic circuits and usually used as digital compasses.

3.1.5 Inertial Sensors

Inertial sensors are sensing devices which are based on inertia. Accelerometers and gyroscopes are two commonly used inertial sensors which rely on microelectromechanical system or laser technology. A MEMS accelerometer consists of mass-spring systems. Displacement of the mass, upon exerting an acceleration, determines the amount of the acceleration. A MEMS gyroscope, used to measure orientation rates,

is composed of a small vibrating mass and a suspended spring system. The mass oscillates at a fixed frequency. When the gyroscope is rotated, gyroscopic effects on the mass provide the rotation rates. MEMS sensors suffer from bias and drift. In contrast, fiber optic gyros (FOG) and ring laser gyroscopes (RLG), which are based on the Sagnac effect, are very reliable. Laser gyroscopes are very accurate and they are often used without any reference sensors. For this reason, laser gyroscopes are very expensive. While MEMS gyroscopes are relatively inexpensive, their low accuracy is compensated for by filtering and reference sensors. Accelerometers, gyroscopes, and magnetometers are commonly available in small packages, called inertial measurement units. IMUs are widely used in robotic applications. Data from these three sensors are fused to provide reliable information.

The CH Robotics UM6 is a lightweight IMU which measures the orientation in 3D and has a built-in filtering solution. It operates at 500 Hz.

3.1.6 Wheel Encoder

A wheel encoder, also called a rotary encoder, is an electro-mechanical sensor that converts the angular position of a rotary shaft to digital or analog codes. Encoders are either incremental or absolute. An absolute encoder shows the current position of the shaft and requires further processing to determine the velocity of the shaft. An incremental encoder, which indicates the motion of the shaft, also requires further processing to determine the position or velocity of the shaft. Optical incremental encoders are widely used in robotic applications to determine the position or velocity of wheels. These encoders generate a certain number of pulses for each revolution of the shaft. Speed of the pulse generation corresponds to the angular speed of the shaft. In a skid-steering robot, wheels on each side of the robot are equipped with a wheel encoder. By processing measurements from encoders on both sides of the robot, position or velocity of the robot is estimated. These estimates are subject to

drift due to slippage of the wheels and must be corrected by other sensors.

3.1.7 Radio-frequency Identification

Radio-frequency identification (RFID) uses radio-frequency electromagnetic fields to transfer data. An RFID tag is a very small circuit which contains electronic information. RFID tags are tracked by radio signals they emit. This feature of RFID is widely used in industry and robotic applications to track or localize objects which have RFID tags. To identify a tag, it should be within an appropriate distance from the tag reader. Often the distance between the tag and the reader is identified according to the power of the wireless signal. The weaker the signal, the further the tag is from the reader. An RFID system requires initial setup; the tag readers should be mounted in advance. The initial setup limits applications of RFID in robotics for mobile robots.

3.1.8 Global Positioning System

The global positioning system, developed in 1973, but not fully operational until the mid 1990s, is a system of satellites orbiting around the Earth which provides location and time information to any device with a GPS receiver. The GPS receiver must be in line-of-sight with four or more GPS satellites. It provides position estimates with bounded error (typically ± 10 m for position estimates); however, the quality of GPS signals is affected in urban areas due to a phenomenon called multi-path. Moreover, it is not available in indoor and underwater environments.

3.1.9 Motion Capture System

A motion capture system, such as Vicon, is a system of multiple cameras used indoors to capture and track motions. To track an object, reflective markers are attached to

it. The positions of the markers are determined through processing multiple images from all cameras. The accuracy of the motion capture positioning is on the order of 1 mm, and a typical motion capture system can provide estimates at the rate of 30 Hz. However, the system requires initial setup and is expensive. Moreover, the working space is limited to the volume visible to the cameras.

3.2 Probabilistic Estimation

In probabilistic robotics, different types of uncertainties are taken into account by a few principle rules of statistics. These key rules are briefly presented in this section. Further information can be found in [3] and [58].

3.2.1 Conditional Probability

Conditional probability states the probability of a random variable X taking a value x , given that another random variable Z takes the value of z . This is stated as

$$p(x|z) = p(X = x|Z = z). \quad (3.1)$$

This probability is formulated as following

$$p(x|z) = \frac{p(x, z)}{p(z)}, \quad (3.2)$$

where it is assumed that $p(z) > 0$ and $p(x, z)$ is the *joint probability* of the two random variables. Fig. 3.1 depicts the Venn diagram of conditional probability.

From the conditional probability, defined in equation (3.2), the *Chain rule* of the joint probability is derived. For example, for two random variables such as $X = x$ and

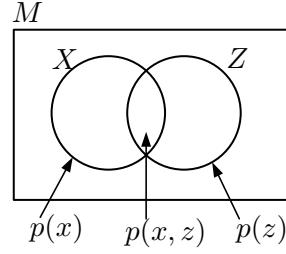


Figure 3.1: Conditional probability, $p(x|z)$. M is the space of all possibilities for the random variables.

$Z = z$, the chain rule is stated as

$$p(x, z) = p(z) p(x|z). \quad (3.3)$$

A conditional probability such as $p(x|z)$ can be conditioned on a third random variable such as Y . For instance, conditioning $p(x|z)$ on $Y = y$ results in [3]

$$p(x|z, y) = \frac{p(z|x, y) p(x|y)}{p(z|y)}. \quad (3.4)$$

This relation is used in this work frequently and is one of the key statistical relations in recursive estimation. To verify equation (3.4), assuming that $p(z, y) \neq 0$, it can be shown that

$$p(x|z, y) = \frac{p(x, z, y)}{p(z, y)} \quad (3.5)$$

$$= \frac{p(z|x, y) p(x, y)}{p(z, y)} \quad (3.6)$$

$$= \frac{p(z|x, y) \overbrace{p(x|y) p(y)}^{p(x,y)}}{\underbrace{p(z|y) p(y)}_{p(z,y)}} \quad (3.7)$$

$$= \frac{p(z|x, y) p(x|y)}{p(z|y)}. \quad (3.8)$$

The Chain rule, stated in equation (3.3), can also be extended to be conditional on

another random variable such as $Y = y$. This is called the *conditional Chain rule* and is stated as

$$p(x, z|y) = p(x|y) p(z|x, y). \quad (3.9)$$

To verify it, assuming that $p(x, y) \neq 0$, it can be shown that

$$p(x, z|y) = \frac{p(x, z, y)}{p(y)} \quad (3.10)$$

$$= \frac{p(x, y)}{p(y)} \frac{p(z, x, y)}{p(x, y)} \quad (3.11)$$

$$= p(x|y) p(z|x, y). \quad (3.12)$$

Conditional independence is another important rule in probabilistic robotics. It is mainly used when a random variable Y carries no information about another random variable X , if the value of another random variable Z is known:

$$p(x, y|z) = p(x|z) p(y|z). \quad (3.13)$$

3.2.2 Bayes Rule

Bayes rule, also called *Bayes theorem*, is the core of most statistical estimations. Followed by conditional probability, the Bayes rule is stated as

$$\underbrace{p(x|z)}_{\text{posterior}} = \frac{\underbrace{p(z|x)}_{\text{likelihood}} \underbrace{p(x)}_{\text{prior}}}{\underbrace{p(z)}_{\text{evidence}}}. \quad (3.14)$$

In the literature $p(z|x)$ is referred to as the *likelihood* and in general it is easier to calculate than calculating $p(x|z)$. For this reason Bayes rule plays a key role in statistical estimation and filtering. $p(x)$ is the probability of $X = x$, before incorporating the random variable Z and therefore is referred to as the *prior* probability. $p(z)$ acts

as *evidence* to evaluate $p(x|z)$. $p(x|z)$ is the probability of $X = x$, after incorporating the random variable Z and therefore is called the *posterior*.

3.2.3 Marginalization

Marginalization, also referred to as the theorem of total probability, is derived from the conditional probability. Marginalization is an extension of the conditional probability which takes into account all possible occurrences of a conditional probability and is defined as

$$p(x) = \int p(x|z) p(z) dz. \quad (3.15)$$

A conditional probability can also be marginalized on a third random variable, $Y = y$, as follows:

$$p(x|z) = \int p(x|z, y) p(y|z) dy. \quad (3.16)$$

3.2.4 Probabilistic Models

In engineering problems which need probabilistic estimation, system models need to be represented probabilistically. For instance, assume a nonlinear system has been modelled by the following nonlinear equations

$$x_t = f(x_{t-1}, u_t) + \delta_t \quad (3.17)$$

$$z_t = h(x_t) + \epsilon_t, \quad (3.18)$$

where $f(\cdot)$ is the nonlinear motion model and $h(\cdot)$ is the nonlinear measurement model. x_t , u_t , and z_t are the state of the system, the control input, and the observation at time t , respectively. δ_t and ϵ_t are the additive process and measurement noises which are Gaussian distributions with means of 0 and covariances of Q and R , respectively. Fig. 3.2 shows how the state, the control, and the measurement at each

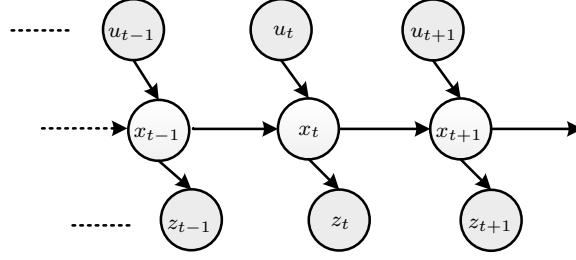


Figure 3.2: State transition diagram or the Bayes net represents the evolution of states, controls, and measurements from time $t - 1$ to $t + 1$.

time evolve. At time t , x_t is derived from x_{t-1} and u_t through the motion model. At the same time, measurement z_t is made given the state, x_t . Because of the noise and other modelling uncertainties, the state at each time is stochastic and represented by a probability distribution function (PDF). The distribution $p(x_t|x_{t-1}, u_t)$ represents the *motion model* in a probabilistic format. $p(x_t|x_{t-1}, u_t)$ is a PDF not a deterministic function, and it shows how the state evolves given the control. The distribution $p(z_t|x_t)$ reflects the probability of making the measurement z_t from the state x_t and is called the probabilistic *measurement model*. In general, these distributions can have any shape; however, for Gaussian state and noise, these probabilistic models can be represented by Gaussian distributions defined as [59]

$$p(x_t|x_{t-1}, u_t) \propto e^{-\frac{1}{2} \|f(x_{t-1}, u_t) - x_t\|_Q^2} \quad (3.19)$$

$$p(z_t|x_t) \propto e^{-\frac{1}{2} \|h(x_t) - z_t\|_R^2}, \quad (3.20)$$

where \propto shows the proportionality of two sides of the equation. For a state vector e with the covariance matrix Σ , $\|e\|_\Sigma^2 \triangleq e' \Sigma^{-1} e$ is the squared Mahalanobis distance.

3.2.5 Bayes Filter

Suppose $x_{1:t}$ represents the state of a system from time 1 to time t , $u_{1:t}$ represents all control actions from time 1 to time t , and $z_{1:t}$ represents all observations from time

1 to time t . Often in the literature the following notations are used for simplicity [3]

$$\overline{\text{bel}}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \quad (3.21)$$

$$\text{bel}(x_t) = p(x_t|z_{1:t}, u_{1:t}), \quad (3.22)$$

where the probabilities are represented by a belief function. The belief in the prior is shown by $\overline{\text{bel}}(x_t)$. This probability represents the probability of x_t before incorporating the latest available measurement, z_t . The belief in the posterior is shown by $\text{bel}(x_t)$. In other words, this probability represents the probability of x_t given the latest measurement, z_t . Probabilities in equations (3.21) and (3.22) are extended as [3]:

$$\overline{\text{bel}}(x_t) = \int p(x_t|x_{t-1}, u_t) \text{bel}(x_{t-1}) dx_{t-1} \quad (3.23)$$

$$\underbrace{\text{bel}(x_t)}_{\text{posterior}} = \eta \underbrace{p(z_t|x_t)}_{\text{likelihood}} \underbrace{\overline{\text{bel}}(x_t)}_{\text{prior}}, \quad (3.24)$$

where in equation (3.24), the *evidence* acts as a normalizer shown by η . Equations (3.23) and (3.24) construct the Bayes filter. Note that equation (3.23) is performed over all possible values of state x_{t-1} .

The general form of the Bayes filter is represented in Algorithm 3.2.1. In line 2, a convolution operation is performed between the posterior belief at time $t - 1$ and the motion model, $p(x_t|x_{t-1}, u_t)$. This step is often referred to as the *prediction* step. In line 3, based on the measurement model, $p(z_t|x_t)$, prior belief is updated. This is called the *update* step. η is a normalizer which makes $\text{bel}(x_t)$ to be a valid probability distribution. Detailed information and the proof of the algorithm can be found in [3]. The Kalman filter, Hidden Markov Models (HMM), and particle filters are different implementations of the Bayes filter.

Algorithm 3.2.1 Bayes filter.

Input: belief at time $t - 1$: $\text{bel}(x_{t-1})$,
control signal at time t : u_t ,
observation at time t : z_t .

Output: belief at time t : $\text{bel}(x_t)$.

- 1: **for all** x_t **do**
- 2: $\overline{\text{bel}}(x_t) \leftarrow \int p(x_t|x_{t-1}, u_t) \text{bel}(x_{t-1}) dx_{t-1}$
- 3: $\text{bel}(x_t) \leftarrow \eta p(z_t|x_t) \overline{\text{bel}}(x_t)$
- 4: **end for**
- 5: return $\text{bel}(x_t)$

3.2.6 Kalman Filter

The Kalman filter is the best known implementation of the Bayes filter for linear systems, assuming additive Gaussian noise for motion and measurement models. The Kalman filter was first proposed in 1960 by Rudolf E. Kalman. Since then it has had many applications in engineering and estimation problems.

Assume the motion and measurement models of a system are represented as follows:

$$x_t = A_t x_{t-1} + B_t u_t + \delta_t \quad (3.25)$$

$$z_t = C_t x_t + \epsilon_t, \quad (3.26)$$

where x_t is the state of the system at time t , u_t is the control input at time t , and z_t is the observation at time t . A_t , B_t , and C_t are system matrices, obtained using system modelling and identification techniques. δ_t is the additive process noise which is a Gaussian noise with a mean of 0 and covariance of Q . ϵ_t is the additive measurement noise which is a Gaussian noise with a mean of 0 and covariance of R . The Kalman filter assumes a Gaussian distribution for the state, x_t , and estimates mean and covariance of the state at time t , *i.e.*, μ_t and Σ_t . This is shown by $\mathcal{N}(x_t; \mu_t, \Sigma_t)$.

Algorithm 3.2.2 represents the Kalman filter. Lines 1 and 2 predict the state of the system based on the motion model given the control input at time t . Line 3 calculates the so called Kalman gain. This gain is used to update the predicted state based on

the measurement signal. Lines 3 and 4 show the update process.

For a linear system, it is proved that the Kalman filter can provide optimal state estimation. It is relatively easy to implement it; however, the dependency of the filtering on the statistics of the noise signals requires special attention in implementations. If mean and covariance of the noise signals are not known properly, the Kalman filter will not provide optimal results.

Algorithm 3.2.2 Kalman filter.

Input: state at time $t - 1$: μ_{t-1}, Σ_{t-1} ,
control signal at time t : u_t ,
observation at time t : z_t .

Output: state at time t : μ_t, Σ_t .

- 1: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
 - 2: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t' + Q$
 - 3: $K_t = \bar{\Sigma}_t C_t' (C \bar{\Sigma}_t C' + R)^{-1}$
 - 4: $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
 - 5: $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
 - 6: return μ_t, Σ_t
-

3.2.7 Extended Kalman Filter

If the system equations are nonlinear, then the Kalman filter cannot be applied directly. The reason for this problem is the covariance matrix. The mean of the state can be calculated using the nonlinear system equations; however, in order to calculate the covariance matrix, nonlinear equations must be linearized. This solution is called the extended Kalman filter (EKF) and is a suboptimal solution for nonlinear systems. Assume the motion and measurement models of a nonlinear system are represented as follows:

$$x_t = f(x_{t-1}, u_t) + \delta_t \quad (3.27)$$

$$z_t = h(x_t) + \epsilon_t, \quad (3.28)$$

where $f(\cdot)$ is the nonlinear motion model and $h(\cdot)$ is the nonlinear measurement model. The rest of the parameters are defined as they are for the Kalman filter. Algorithm 3.2.3 represents the EKF. In line 1, the mean of the state is predicted using the nonlinear motion model. In line 2, the covariance of the predicted state is calculated. Note that F_t is the Jacobian of the motion model defined as

$$F_t = \frac{\partial f(x_{t-1}, u_t)}{\partial x_{t-1}} \Big|_{\mu_t} \quad (3.29)$$

Similar to the Kalman filter, in lines 3 – 5, the predicted state is updated, except that the matrix C_t in Algorithm 3.2.2 is replaced with H_t which is the Jacobian of the measurement model, defined as

$$H_t = \frac{\partial h(x_t)}{\partial x_t} \Big|_{\bar{\mu}_t} \quad (3.30)$$

The EKF is a suboptimal solution. Similar to the Kalman filter, it is relatively easy to implement; however, linearization of the system equations may cause problems at the points where the behaviour of the system is highly nonlinear.

Algorithm 3.2.3 Extended Kalman filter.

Input: state at time $t - 1$: μ_{t-1} , Σ_{t-1} ,
control signal at time t : u_t ,
observation at time t : z_t .

Output: state at time t : μ_t , Σ_t .

- 1: $\bar{\mu}_t = f(\mu_{t-1}, u_t)$
 - 2: $\bar{\Sigma}_t = F_t \Sigma_{t-1} F_t' + Q$
 - 3: $K_t = \bar{\Sigma}_t H_t' (H \bar{\Sigma}_t H' + R)^{-1}$
 - 4: $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$
 - 5: $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
 - 6: return μ_t , Σ_t
-

3.2.8 Information Filter

The information filter is another efficient implementation of the Bayes filter. In fact it is known to be the dual of the Kalman filter. The Kalman filter estimates the mean and the covariance of the state while the information filter estimates the canonical parameters which are the information vector and the information matrix, defined as

$$\Omega = \Sigma^{-1} \quad (3.31)$$

$$\zeta = \Sigma^{-1} \mu, \quad (3.32)$$

where μ and Σ are the mean and covariance, and ζ and Ω are the information vector and the information matrix. For the system defined in equation (3.25), the information filter is presented in Algorithm 3.2.4. Lines 1 and 2 predict the state of the system using the motion model. Lines 3 and 4 update the predicted state using the measurement model. The information filter has the advantage that the update step does not require any matrix inversion; however, the matrix inversion is required in prediction. Similar to the EKF, the information filter can also be extended to be applied to nonlinear systems, which is known as the extended information filter (EIF).

Algorithm 3.2.4 Information filter.

Input: information at time $t - 1$: ζ_{t-1} , Ω_{t-1} ,
control signal at time t : u_t ,
observation at time t : z_t .

Output: information at time t : ζ_t , Ω_t .

- 1: $\bar{\Omega}_t = (A_t \Omega_{t-1}^{-1} A_t' + Q)^{-1}$
 - 2: $\bar{\zeta}_t = \bar{\Omega}_t (A_t \Omega_{t-1}^{-1} \zeta_{t-1} + B_t u_t)$
 - 3: $\Omega_t = C_t' Q_t^{-1} C_t + \bar{\Omega}_t$
 - 4: $\zeta_t = C_t' Q_t^{-1} z_t + \bar{\zeta}_t$
 - 5: return ζ_t , Ω_t
-

3.2.9 Particle Filter

Particle filters approximate the posterior of the state by a finite and random set of states. Unlike the EKF and the EIF, a particle filter does not linearize the system equations and therefore is a suitable solution for nonlinear systems. However, its space complexity grows exponentially with respect to the dimension of the state variable. Therefore special modifications are required to make an efficient implementation. Fig. 3.3 shows the Bayes net of the transition of the state given control and measurements.

Particle filtering can be summarized in three main steps:

- **Sampling:** This step generates the next generation of particles from the previous generation. Usually the states of the selected particles are predicted using a motion model.
- **Importance Weighting:** Each particle holds an importance weight, denoting the similarity of the proposal distribution to the target distribution. It is defined as

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)} | z_{1:t}, u_{1:t}, x_0)}{\pi(x_{1:t}^{(i)} | z_{1:t}, u_{1:t}, x_0)}, \quad (3.33)$$

where $p(\cdot)$ is the posterior distribution which is referred to as the target distribution and $\pi(\cdot)$ is the proposal distribution (note that equation (3.33) is defined

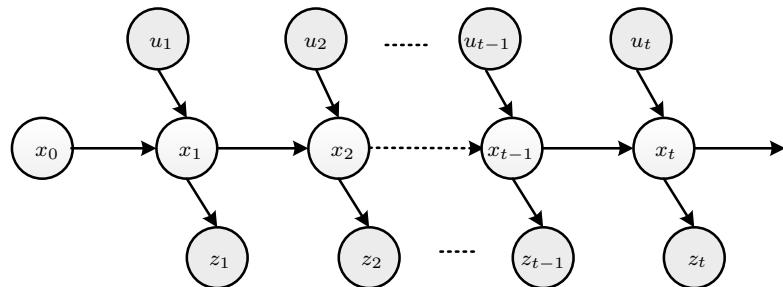


Figure 3.3: Bayes net for a particle filter. Control signal from time 1 to time t , $u_{1:t}$, and measurements from time 1 to time t , $z_{1:t}$ are used to estimate the state of the system from time 1 to time t , $x_{1:t}$. x_0 is the initial state of the system.

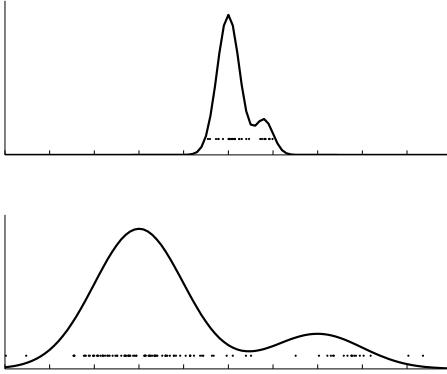


Figure 3.4: Generally, for a peaked distribution (top) fewer particles are required than for a highly uncertain distribution (bottom).

for full trajectory estimation and is in the general form; later the Markov assumption will be applied to it). The proposal distribution is a distribution which is thought to be close to the target distribution. Particle weights are updated based on the similarity of the proposal and target distributions. Since there is no direct access to the target distribution, it is important to have a good proposal distribution. Choosing a good proposal distribution is very important for the efficiency of the filter. A good proposal would be one with less uncertainty. The more uncertain the proposal distribution, the more particles are required to represent the distribution. Fig. 3.4 shows this concept. For the peaked distribution¹ only 20 particles suffice to represent it, while for the other distribution which is not peaked, 100 particles are needed to represent the distribution. In robotic applications, especially with sensors such as laser rangers which provide peaked measurement distributions, by choosing a peaked proposal, space complexity can be reduced.

- **Resampling:** To avoid particle degeneration, resampling is performed. This process behaves in a similar way to the survival of the fittest in the artificial

¹A peaked distribution is one that when approximated by a Gaussian distribution has relatively small 3σ boundaries. In other words, a peaked distribution is less uncertain.

intelligence algorithms. But frequent resampling can also cause degeneracy problems. Calculating the effective sample size, which is a measure of the dispersion of the importance weights, can indicate when to resample. For M particles, this measure is defined as [60]

$$N_{eff} = \frac{1}{\sum_{i=1}^M (\hat{w}^{(i)})^2} \quad (3.34)$$

where $\hat{w}^{(i)}$ is the normalized weight of the i^{th} particle. N_{eff} varies between 1 and M . Doucet *et al.* [61] showed that to avoid the degeneracy problem, the best time to resample is when N_{eff} falls below $M/2$.

With this three-step particle filter, there is a key limitation in implementation. Whenever a new observation is made, weights of trajectories should be updated from scratch. Since the length of the trajectory is increasing over time, this filtering would fail. Doucet *et al.* [61] showed that by choosing the following recursive proposal distribution, it is possible to have a recursive weight-update.

$$\pi(x_{1:t}|z_{1:t}, u_{1:t}) = \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}). \quad (3.35)$$

This equation can also be derived from the conditional chain rule shown in equation (3.9) as follows:

$$\pi(x_{1:t}|z_{1:t}, u_{1:t}) = \pi(\underbrace{x_{1:t-1}}_a, \underbrace{x_t}_b | \underbrace{z_{1:t}, u_{1:t}}_c) \quad (3.36)$$

$$\stackrel{Eq.(3.9)}{=} \pi(\underbrace{x_{1:t-1}}_a | \underbrace{z_{1:t}, u_{1:t}}_c) \pi(\underbrace{x_t}_b | \underbrace{x_{1:t-1}, z_{1:t}, u_{1:t}}_c) \quad (3.37)$$

$$= \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}) \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}), \quad (3.38)$$

where a , b , and c are three parameters arbitrarily chosen to show how the conditional

Chain rule, equation (3.9), is applied. By placing equation (3.35) in equation (3.33) and applying the basic rule of conditioning on a third random variable, equation (3.4), we will have the recursive weight-update [3, 43]:

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)} | z_{1:t}, u_{1:t}, x_0)}{\pi(x_{1:t}^{(i)} | z_{1:t}, u_{1:t}, x_0)} \quad (3.39)$$

$$= \frac{p(x_{1:t}^{(i)} | z_t, z_{1:t-1}, u_{1:t}, x_0)}{\pi(x_{1:t}^{(i)} | z_{1:t}, u_{1:t}, x_0)} \quad (3.40)$$

$$= \frac{p(\overbrace{x_{1:t}^{(i)}}^{\text{a}} | \overbrace{z_t}^{\text{b}}, \overbrace{z_{1:t-1}, u_{1:t}, x_0}^{\text{c}})}{\pi(x_t^{(i)} | x_{1:t-1}, z_{1:t}, u_{1:t}, x_0) \pi(x_{1:t-1}^{(i)} | z_{1:t-1}, u_{1:t-1}, x_0)} \quad (3.41)$$

$$\stackrel{\text{Eq.(3.4)}}{=} \frac{p(\overbrace{z_t}^{\text{b}} | \overbrace{x_{1:t}^{(i)}}^{\text{a}}, \overbrace{z_{1:t-1}, u_{1:t}, x_0}^{\text{c}}) p(\overbrace{x_{1:t}^{(i)}}^{\text{a}} | \overbrace{z_{1:t-1}, u_{1:t}, x_0}^{\text{c}})}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t}, x_0) \pi(x_{1:t-1}^{(i)} | z_{1:t-1}, u_{1:t-1}, x_0)}. \quad (3.42)$$

The probability $\frac{1}{p(z_t | z_{1:t-1}, u_{1:t}, x_0)}$ has the same value for all particles; therefore it is treated as a normalizer and replaced by η . Thus, the weight of the i^{th} particle is

$$w_t^{(i)} = \eta \frac{p(z_t | x_{1:t}^{(i)}, z_{1:t-1}, u_{1:t}, x_0) p(x_{1:t}^{(i)} | z_{1:t-1}, u_{1:t}, x_0)}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t}, x_0) \pi(x_{1:t-1}^{(i)} | z_{1:t-1}, u_{1:t-1}, x_0)} \quad (3.43)$$

$$\stackrel{\text{Eq.(3.10)}}{=} \eta \frac{p(z_t | x_{1:t}^{(i)}, z_{1:t-1}, x_0) p(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t-1}, u_{1:t}, x_0)}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t}, x_0)} \cdot \frac{p(x_{1:t-1}^{(i)} | z_{1:t-1}, u_{1:t-1}, x_0)}{\pi(x_{1:t-1}^{(i)} | z_{1:t-1}, u_{1:t-1}, x_0)} \quad (3.44)$$

$$= \eta \frac{p(z_t | x_{1:t}^{(i)}, z_{1:t-1}, x_0) p(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t-1}, u_{1:t}, x_0)}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t}, x_0)} w_{t-1}^{(i)} \quad (3.45)$$

$$\stackrel{\text{Markov}}{=} \eta \frac{p(z_t | x_t^{(i)}) p(x_t^{(i)} | x_{t-1}^{(i)}, u_t)}{\pi(x_t^{(i)} | x_{t-1}^{(i)}, u_t, z_t)} w_{t-1}^{(i)} \quad (3.46)$$

Therefore, weights are recursively updated according to the following relation, where \propto shows proportionality of two sides of the equation.

$$w_t^{(i)} \propto \frac{p(z_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)}, u_t)}{\pi(x_t^{(i)}|x_{t-1}^{(i)}, u_t, z_t)} w_{t-1}^{(i)}. \quad (3.47)$$

Notice that in implementation, there is no need to calculate the normalizer, η , directly, since it has a constant value for all particles. However, to consider the effect of the normalizer, a feasible solution is to calculate all weights without the normalizer, and then scale them such that the sum of all weights is one. As a result, in probabilistic robotics, usually equations with normalizers are replaced with two proportional terms without any normalizer, as performed in equation (3.47).

By choosing the motion model for the proposal $\pi(\cdot) = p(x_t|x_{t-1}, u_t)$ in (3.47), the weight-update becomes:

$$w_t^{(i)} \propto p(z_t|x_t^{(i)}) w_{t-1}^{(i)} \quad (3.48)$$

This equation simply states that weights are updated based on the measurement model, given that the motion model is used as the proposal distribution.

Algorithm 3.2.5 summarizes the particle filter. Line 4 performs sampling which is equivalent to the *prediction* in the Bayes filter. The sign \sim in this line shows that $x_t^{(i)}$ is sampled according to the distribution $p(x_t|x_{t-1}^{(i)}, u_t)$. The sampling is based on the motion model. Line 5 updates weights based on the measurement following the relation presented in equation (3.48). In line 6, the updated particles are added to the particle set of the posterior. If the dispersion of the importance weights, N_{eff} , is less than $M/2$, then resampling proportional to the weights of the particles is performed (line 10). Function `resample()` does this operation.

Algorithm 3.2.5 Particle filter.

Input: Set of particles representing posterior at time $t - 1$: S_{t-1} ,
control signal at time t : u_t ,
observation at time t : z_t .

Output: Set of particles representing posterior at time t : S_t .

```
1:  $S_t \leftarrow \emptyset$ 
2: for  $i = 1 \rightarrow M$  do
3:    $\langle x_{t-1}^{(i)}, w_{t-1}^{(i)} \rangle \leftarrow S_{t-1}^{(i)}$ 
4:   sample  $x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_t)$ 
5:    $w_t^{(i)} \leftarrow p(z_t | x_t^{(i)}) w_{t-1}^{(i)}$ 
6:    $S_t \leftarrow S_t \cup \langle x_t^{(i)}, w_t^{(i)} \rangle$ 
7: end for
8: if  $N_{eff} < \frac{M}{2}$  then
9:   for  $i = 1 \rightarrow M$  do
10:    resample( $S_t$ )
11:   end for
12: end if
13: return  $S_t$ 
```

3.2.10 Least Squares Regression

In robotics, least squares regression, also called smoothing or maximum *a posteriori* (MAP), is a mathematical optimization method which minimizes the sum of squared motion and measurement constraints. It is widely used for nonlinear systems. Consider the nonlinear system presented in equations (3.49) and (3.50).

$$x_t = f(x_{t-1}, u_t) + \delta_t \quad (3.49)$$

$$z_t = h(x_t) + \epsilon_t, \quad (3.50)$$

where $f(\cdot)$ is the nonlinear motion model and $h(\cdot)$ is the nonlinear measurement model. x_t , u_t , and z_t are the state of the system, the control input, and the observation at time t . $x_{1:t}$, $z_{1:t}$, $u_{1:t}$ represent the state of the system, the observations, and the control signals from time 1 to t , respectively. x_0 is the initial state of the system. δ_t and ϵ_t are the additive process and measurement noises which are Gaus-

sian distributions with means of 0 and covariances of Q and R , respectively. The smoothing is defined as finding the entire trajectory, $x_{1:t}^*$, through minimizing the following equation [59]:

$$x_{1:t}^* = \underset{x_{1:t}}{\operatorname{argmin}} \left(\sum_{k=1}^t \frac{1}{2} \|f(x_{k-1}, u_k) - x_k\|_Q^2 + \sum_{k=1}^t \frac{1}{2} \|h(x_k) - z_k\|_R^2 \right). \quad (3.51)$$

If the process model $f(\cdot)$ and measurement model $h(\cdot)$ are nonlinear and there is no good linearization point, then nonlinear optimization solutions such as the Levenberg-Marquardt method or Gauss-Newton approach finds $x_{1:t}^*$ by solving a succession of linear approximations to equation 3.51.

The presented state estimation algorithms are summarized in Table 3.1. More details can be found in [3].

3.3 Background on Single-robot SLAM

In this section, different techniques of SLAM for one robot are reviewed briefly. Most of these techniques are based on the filtering methods introduced in the previous section. Before introducing these methods, localization and mapping as two independent problems are briefly reviewed.

3.3.1 Localization

Localization is the problem of estimating the pose of a robot, given the map of the environment, control actions taken by the robots and observations made by the robot from the environment. Fig. 3.5 shows the Bayes net for the localization problem. In this figure, shaded circles represent known states while unshaded circles indicate unknown states. Knowledge of x_0 , the initial pose of the robot, distinguishes two general cases for localization. At times $1, 2, \dots, t$, let $x_{1:t}$ and $z_{1:t}$ denote corresponding

Table 3.1: Some general methods for state estimation

State Estimator	Description
Bayes Filter	<p>Filtering is performed recursively given system equations</p> <p>Pros: optimal, multi-modal</p> <p>Cons: computationally intractable</p>
Kalman Filter	<p>State distribution assumed to be Gaussian and parameterized by mean, μ, and covariance, Σ</p> <p>Pros: optimal if the system is linear and the noise and system states are Gaussian</p> <p>Cons: requires system equations to be linear, needs tuned noise statistics</p>
Extended Kalman Filter	<p>Extension of the Kalman filter for nonlinear systems</p> <p>Pros: works well if the system is not highly nonlinear</p> <p>Cons: needs system linearization, needs tuned noise statistics, measurement update is slow as it needs matrix inversion</p>
Extended Information Filter	<p>Dual of the extended Kalman filter, state distribution assumed Gaussian shown by canonical parameters</p> <p>Pros: measurement update is fast, allows processing multiple measurements at one time through addition</p> <p>Cons: needs linearization, needs tuned noise statistics, prediction is slow as it needs matrix inversion, time consuming to recover the mean and covariance</p>
Particle Filter	<p>Represents the state distribution by a finite set of weighted particles</p> <p>Pros: supports non-Gaussian distributions and nonlinear models</p> <p>Cons: computationally expensive</p>
Least Squares Regression	<p>Also called smoothing, minimizes the sum of squared motion and measurement constraints</p> <p>Pros: past states are maintained which can be useful for full state trajectory estimation</p> <p>Cons: needs system linearization</p>

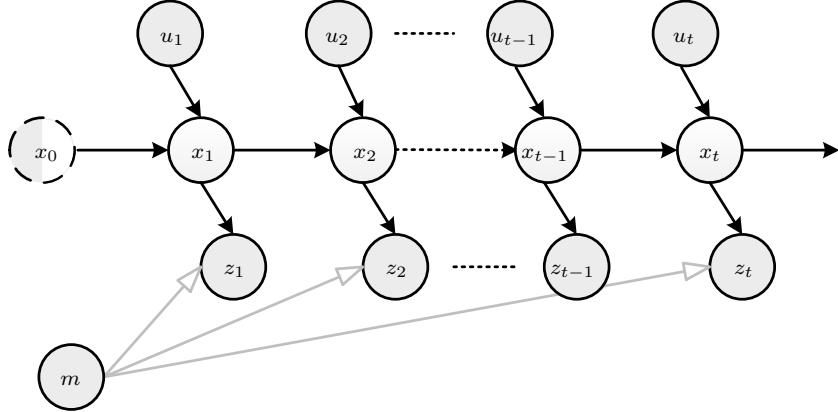


Figure 3.5: Bayes net for localization. Solid black lines correspond to the state flow of each individual robot. Grey lines show the relation of the map and observations. Shaded circles are known states; unshaded circles are unknown states. Knowledge of x_0 , shown with a broken perimeter, identifies two types of the localization problem.

sequences of poses and measurements, respectively. The sequence of action signals is given by $u_{1:t}$. Localization seeks to calculate the posterior over the trajectory of the robot, given the map, the action signals, measurement signals, and possibly the initial pose of the robot:

$$p(x_{1:t}|m, z_{1:t}, u_{1:t}, x_0). \quad (3.52)$$

If the initial pose is known, the localization problem is referred to as *position tracking*, or *local localization*, or just *localization*. But if it is unknown, the problem is called *global localization*. Many solutions are available for these problems. Most are based on filtering approaches introduced in the previous section. In general, the pose of the robot, which includes position and orientation, needs to be estimated using the available information. More details can be found in [3].

3.3.2 Mapping

Mapping is the problem of estimating the map of the environment, given the pose of the robot, and observations made by the robot from the environment. Fig. 3.6 shows the Bayes net for the mapping problem. In this figure, shaded circles depict known

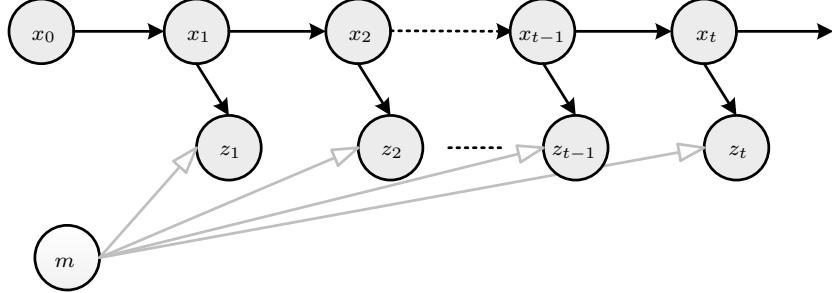


Figure 3.6: Bayes net for mapping. Solid black lines correspond to the state flow of each individual robot. Grey lines show the relation of the map and observations. Shaded circles are known states; unshaded circles are unknown states.

states while unshaded circles indicate unknown states. At times $1, 2, \dots, t$, $x_{1:t}$ and $z_{1:t}$ show corresponding sequences of poses and measurements, respectively.

A key issue that distinguishes different mapping and therefore different methods of SLAM, is the map representation. Generally, four types of maps are commonly used: grid maps, feature maps, topological maps, and hybrid maps.

Grid maps, also called *location maps*, represent the environment with a matrix of cells. In occupancy grid maps (OGM) which are very popular in 2D mapping, each cell represents a small rectangular section of the world. It is modelled with a binary random variable which indicates the probability of existence of an object in the cell. Later this method is explained in more detail. Occupancy grid maps can also be extended to be 3D and these are called volumetric pixel (voxel) maps.

Feature maps, also called *landmark maps*, represent the world with global position of features, extracted from the environment. Often, the position of the feature is accompanied with the signature of the feature. The signature of a feature is the unique identifier, such as its colour, which is used to distinguish it from others.

Topological maps represent an abstract model of the environment in the form of compact and connected paths and intersections. Humans and insects use topological maps to navigate, argue about paths and positions and avoid obstacles [62, 63]. As an example, pigeons have been shown to use highways and their intersections as a

Table 3.2: Comparison of different mapping methods

Type of Map	Description
Occupancy Grid Maps	Map is defined as grids, each grid holds a probability for occupancy Pros: probabilistic, suitable for 2D mapping, suitable for dynamic environments Cons: expensive on fine resolution, expensive for 3D mapping
Feature Maps	Map is represented by features Pros: efficient for localization, scales well Cons: needs feature extraction, data association is difficult
Topological Maps	Map is represented by abstract information Pros: well suited for high-level planning Cons: needs map processing, limited in waypoint following
Hybrid Maps	Map is a combination of topological and metric maps Pros: suitable for loop closure, can handle map inconsistency Cons: needs map processing, requires coordination between maps

topological map to navigate and fly over long distances [64]. Topological approaches for perception or autonomy can also be interpreted as symbolic mapping [65]. Symbolic approaches provide a concise representation for the structural alternatives which can be used in path planning, place detection, and loop closure. Topological maps can be derived from metric maps. Voronoi graph [66] is a topological map that can be computed from occupancy grid maps. Polygon maps are another example which assign planes to a set of features.

Hybrid maps are combinations of different mapping methods. For example a hybrid topological map is a framework which includes both topological and metric information.

Table 3.2 summarizes these methods with their advantages and disadvantages. In the rest of the section, occupancy grid mapping, which is frequently used throughout this work, is introduced in more detail.

Occupancy grid mapping is a standard method for mapping using range measure-

ments. A map m can be represented as a set of N grid cells, each with an associated binary random variable, representing its occupancy:

$$m = \{m_i\}, i = 1, \dots, N. \quad (3.53)$$

The binary random variable specifies whether a cell is occupied ('1') or free ('0'), then $p(m_i = 1)$ or $p(m_i)$ represents the probability that the cell is occupied.

In an occupancy grid map, it is desired to calculate the posterior over the map given the trajectory of the robot and available measurements as follows [3]:

$$p(m|z_{1:t}, x_{1:t}) = \prod_{i=1}^N p(m_i|z_{1:t}, x_{1:t}) \quad (3.54)$$

The factorization defined in equation (3.54) makes updating the map tractable.

Assuming initially a cell i has *unknown* occupancy, $p(m_i) = 0.5$, at time t ; if the cell is in the perception field of the range sensor, its value is calculated using the following recursive relation:

$$l_{t,i} = l_{t-1,i} + \log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)}, \quad (3.55)$$

where $l_{t,i}$ is referred to as the *log odds* and is defined as

$$l_{t,i} = \log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}. \quad (3.56)$$

The *log odds* representation avoids numerical instabilities for probabilities near zero or one. The probability of $p(m_i|z_t, x_t)$ is sensor specific and for a laser rangefinder can be modelled using the approach presented in [3].

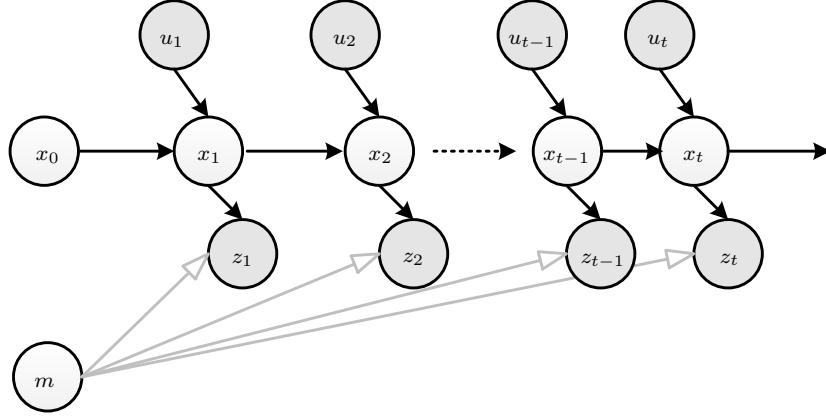


Figure 3.7: Bayes net for single-robot SLAM. Solid black lines correspond to the state flow of each individual robot. Grey lines show the relation of the map and observations.

3.3.3 Definition of Single-robot SLAM

Single-robot SLAM is defined as follows. According to Fig. 3.7, which shows the Bayes net for a single-robot SLAM, assume x_0 is the initial pose of the robot at time 0. At times $1, 2, \dots, t$, let $x_{1:t}$ and $z_{1:t}$ denote corresponding sequences of poses and measurements, respectively. The sequence of action signals is given by $u_{1:t}$. The goal for SLAM is to calculate the posterior over the map, m , and the trajectory given the action signals, measurement signals, and the initial pose of the robot:

$$p(m, x_{1:t} | z_{1:t}, u_{1:t}, x_0). \quad (3.57)$$

Equation (3.57) shows that estimating the map and trajectory of the robot is a coupled problem, which means both must be estimated at the same time. In the literature, this definition of SLAM is also referred to as *full SLAM*, where the whole trajectory is estimated, whereas *online SLAM* involves estimating the posterior over the current pose, x_t , and the map m

$$p(m, x_t | z_{1:t}, u_{1:t}, x_0). \quad (3.58)$$

SLAM problem can be classified into four general cases: *feature-based SLAM*, *view-*

based SLAM, *appearance-based SLAM*, and *polygon-based SLAM*. The rest of this section defines these four cases. In the next sections, different solutions to calculate the posterior in equation 3.57 are presented.

3.3.3.1 Feature-based SLAM

Feature-based SLAM was the first solution proposed for SLAM. This solution extracts features or landmarks in the environment and keeps a list of them as a map [67, 68]. This method requires feature extraction and is therefore limited to environments with features. Most feature-based solutions use vision sensors [69], underwater sonar [70, 71], or occasionally LIDAR [12].

In feature-based SLAM, the state vector, also called the combined state vector, is composed of the pose of the robot and the map (all features). For N features, this state, shown with y_t , is defined as [3]

$$\begin{aligned} y_t &= \begin{pmatrix} x_t \\ m \end{pmatrix} \\ &= \begin{pmatrix} x & y & \theta & m_{1,x} & m_{1,y} & s_1 \dots m_{N,x} & m_{N,y} & s_N \end{pmatrix} \end{aligned} \quad (3.59)$$

where x_t is the pose of the robot at time t which includes position and orientation of the robot, x , y , and θ ; $m_{i,x}$ and $m_{i,y}$ are the coordinates of the i^{th} landmark, for $i = 1, \dots, N$; and s_i is the signature of the i^{th} landmark. Signature of a landmark, as an optional parameter, is the unique identifier of the landmark, such as its colour, which helps distinguish it from others. Different filtering solutions or smoothing approaches are used to estimate this state vector, given control and measurement signals. These methods are reviewed in the following subsections.

While the feature-based paradigm has been shown to be efficient and effective in certain environments, in general it is not. By extracting features, there is possible loss

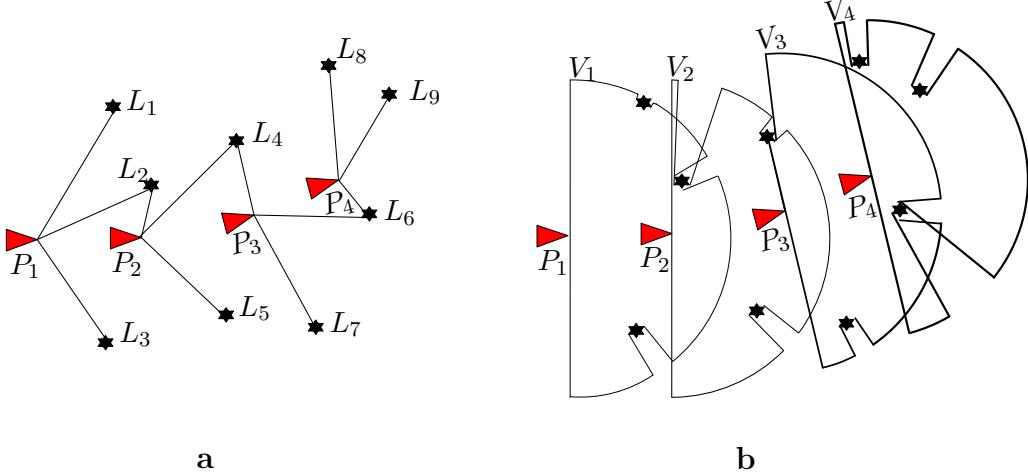


Figure 3.8: P_1 to P_4 are poses of a robot at four time instances. (a) Feature-based SLAM. L_1 to L_9 are detected features used for localization. (b) View-based SLAM. V_1 to V_4 are corresponding views. No features are extracted, instead views are matched for localization [75].

of data. Thrun *et al.* state that “a lot of information is sacrificed by using features instead of full measurement vector. This lost information makes certain problems more difficult, such as the data association problem of determining whether or not the robot just revisited a previously explored location” [3]. Since onboard processing has become so powerful, there is no longer a need to extract features from sensor data and potentially lose useful information in the process. This applies especially to dense range sensors such as laser rangers.

3.3.3.2 View-based SLAM

View-based SLAM, also called location-based SLAM, is based on raw sensor data processing techniques. View-based SLAM usually requires a scanning laser ranger (LIDAR) [43, 72].

View-based SLAM differs from feature-based SLAM in that no features are explicitly extracted [72]. Instead, entire scans are matched using scan matching algorithms [47, 73, 74], (Fig. 3.8) and the map is represented in occupancy grid format [4, 5]. The state vector in view-based SLAM includes either the current pose of the robot or the whole

trajectory. A map also accompanies the trajectory. The most important advantages of view-based SLAM over feature-based SLAM are that since no feature is extracted, no information is lost; also view-based SLAM can handle dynamic environments much better than feature-based SLAM [3, 76].

3.3.3.3 Appearance-based SLAM

Appearance-based SLAM is an effective solution for the loop closure problem [19, 77]. It also can be used in combination with feature-based or view-based SLAM; however, in the literature, appearance-based SLAM is often performed with a camera [18]. In this solution, new observations, either features or views, are matched against available reference observations to identify a previously observed location.

For instance, the ATLAS framework [76] by Bosse *et al.* provides occupancy grid maps from outdoor urban environments using a scanning laser ranger mounted on top of a car. In this solution, salient characteristics of maps are used to develop an appearance recognition system which recalls an already visited location. In [78] position-invariant robust feature (PIRF) detection is used to detect features of images. These features are matched against reference features and used to recall previous scenes.

Due to the necessity of matching a large number of views with reference views, this solution creates high computational demand and often requires a fast processor and relatively large memory to store all views.

3.3.3.4 Polygon-based SLAM

Polygon-based SLAM is often used for 3D mapping [79]. In this method planar segments, composed of infinite planes, are associated with features of the environment. For example, in [80], Weingarten *et al.* perform a 3D SLAM with a robot equipped with a tilting scanning laser ranger. Using polygon sets, features are represented with planar segments. Pathak *et al.* [81] propose a 3D GraphSLAM (see Section 3.3.5) by

registering large planar segments, extracted from a 3D laser ranger. Expectation maximization has also been used for planar mapping by Thrun *et al.* [82]. The advantage of the polygon-based SLAM is that it produces highly detailed small-sized maps and suits well for higher-level tasks such as interacting with the environment; however, the process of fitting planar segments creates extra computational demand.

3.3.4 Filtering Solutions for SLAM

Generally, SLAM solutions are either based on filtering or smoothing techniques. Particle filters, different variations of the Kalman filter and the information filter are major filtering solutions which are derived from the Bayes filter. These techniques are used to estimate the current pose of the robot or to optimize the whole trajectory of the robot. Smoothing methods, also known as GraphSLAM, estimate the whole trajectory of the robot by minimizing the process and observation constraints. Some approaches use a combination of different solutions. For instance, FastSLAM employs both the extended Kalman filter and the particle filter to estimated the trajectory of the robot. There exist other solutions based on artificial intelligence, such as neural networks.

3.3.4.1 EKF-SLAM

In EKF-SLAM, the nonlinear motion and measurement models are linearized using the Taylor expansion. Linearized models are used to predict and update the state recursively. As mentioned, the state of the EKF-SLAM includes the pose of the robot and map features [1, 83]. Therefore, as the robot explores more features, the size of the state grows. Since EKF-SLAM possesses quadratic update complexity, for large maps, EKF-SLAM becomes computationally expensive. Also when the nonlinearity of the motion model increases, for example in sharp turns, the linearized system becomes too approximate to model the motion; however, its simplicity makes it one

of the most well-known solutions for SLAM.

3.3.4.2 EIF-SLAM

The extended information filter has two main variations in SLAM: Sparse Extended Information Filter (SEIF) [84] and Exactly Sparse Extended Information Filter (ESEIF) [85]. In principle, they are both similar to EKF-SLAM: they are recursive and require linearization of the system models. In both solutions, information matrices are actively sparsified. But in ESEIF, the information matrix is sparsified in such a way that the majority of elements are *exactly* zero.

The extended information filter has the advantage over the extended Kalman filter that it scales linearly with the number of dimensions of the state. Therefore, it is a suitable solution for large scale maps; however, accessing the mean and covariance, to recover the map and the pose of the robot for path planning or navigation purposes, requires a large matrix inversion.

3.3.4.3 PF-SLAM

Particle filtering is a nonlinear filtering solution for nonlinear systems; therefore, the motion and measurement models are not linearized or approximated. Compared to the EKF and the EIF, for the SLAM problem, a straightforward implementation of a particle filter will definitely perform badly due to the large size of the state vector [3]; however, a proper implementation such as FastSLAM [86] and the improved grid mapping [43] can reduce its complexity.

FastSLAM is based on a variant of particle filtering known as the Rao-Blackwellized particle filter (RBPF). The RBPF is suitable for systems with a state vector of many random variables. In this filtering, particles represent posterior over some of the variables along with parametric distributions for all other variables. The key idea which allows us to use RBPF is the conditional independency of locations of features,

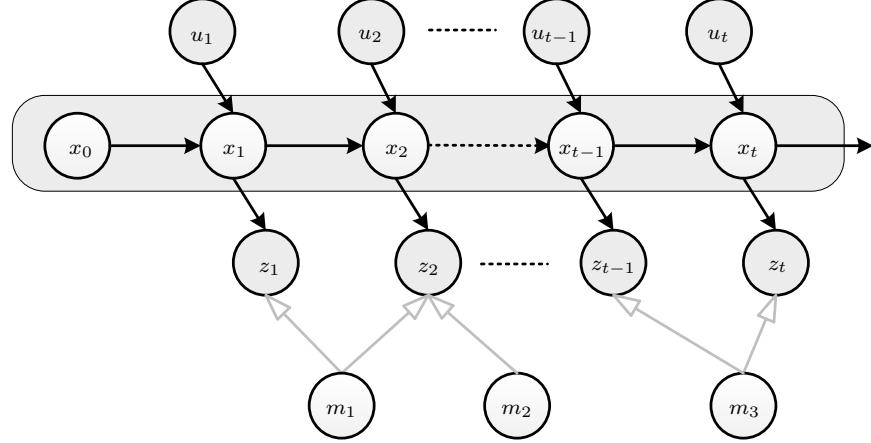


Figure 3.9: Bayes net for a Rao-Blackwellized PF-SLAM. While the robot moves from pose x_0 to x_t , it observes the environment through measurements. If these poses are known, as in the mapping problem, features in the map become conditionally independent.

if the trajectory of the robot is known. This concept is shown in Fig. 3.9. As the figure shows, the knowledge of the poses renders features conditionally independent. In other words, dependencies between measurements arise only through the uncertainty of the robot's poses [3]. Using this fact, the posterior over poses and the map (with N features with known correspondence) is factored as follows:

$$p(m, x_{1:t} | z_{1:t}, u_{1:t}, x_0) = p(m | z_{1:t}, x_{1:t}, x_0) p(x_{1:t} | z_{1:t}, u_{1:t}) \quad (3.60)$$

$$= \prod_{n=1}^N p(m_n | z_{1:t}, x_{1:t}, x_0) p(x_{1:t} | z_{1:t}, u_{1:t}). \quad (3.61)$$

By this factorization, a particle filter is used for calculating the posterior over poses $p(x_{1:t} | z_{1:t}, u_{1:t})$, while calculating the posterior over the location of map features, $p(m_n | z_{1:t}, x_{1:t}, x_0)$, is the problem of mapping with known poses. This technique of factorization is referred to as Rao-Blackwellization.

To construct the filtering, a particle filter is used to estimate the path of the robot. Each particle holds the path of the robot. For each of these particles, a map is also constructed. In feature-based FastSLAM, locations of features are estimated by

EKFs; however, each feature possesses a separate and low-dimension EKF. If there are M particles and N features, the i^{th} particle, $i = 1, 2, \dots, M$, is shown as

$$< x_{1:t}^{[i]}, \underbrace{\mu_1^{[i]}, \Sigma_1^{[i]}, \mu_2^{[i]}, \Sigma_2^{[i]}, \dots, \mu_N^{[i]}, \Sigma_N^{[i]} }_{\text{map}} >, \quad (3.62)$$

where $\mu_j^{[i]}$ and $\Sigma_j^{[i]}$ are the mean and covariance of the position of the j^{th} landmark, $j = 1, 2, \dots, N$. This structure makes FastSLAM fundamentally different from other algorithms like EKF-SLAM, in which a single multivariate Gaussian is used to estimate locations of all features. FastSLAM can be thought of as a particle filter, composed of many Kalman filters.

If a feature is visited for the first time, it is simply added to all particles. Whenever a feature is revisited, it is updated by an EKF.

FastSLAM has the advantage that it is the only solution which performs online SLAM and full SLAM together, which means it estimates not only the current pose, but also the full trajectory. FastSLAM has been introduced in two releases; where FastSLAM2.0 improves upon FastSLAM1.0 by reducing the number of particles and thereby decreasing the complexity. Similar to other methods, FastSLAM is also applicable to view-based SLAM [43], where the map of each particle in equation (3.62) is replaced by an occupancy grid map.

In the full SLAM problem, the weight-update in particle filtering, as defined in equation (3.45), is written as

$$w_t^{(i)} = \eta \frac{p(z_t | x_{1:t}^{(i)}, z_{1:t-1}, x_0) p(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t-1}, u_{1:t}, x_0)}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t}, x_0)} w_{t-1}^{(i)} \quad (3.63)$$

$$\stackrel{\text{Markov}}{=} \eta \frac{p(z_t | x_t^{(i)}, m_{t-1}^{(i)}) p(x_t^{(i)} | x_{t-1}^{(i)}, u_t)}{\pi(x_t^{(i)} | x_{t-1}^{(i)}, u_t, z_t)} w_{t-1}^{(i)} \quad (3.64)$$

$$\propto \frac{p(z_t | x_t^{(i)}, m_{t-1}^{(i)}) p(x_t^{(i)} | x_{t-1}^{(i)}, u_t)}{\pi(x_t^{(i)} | x_{t-1}^{(i)}, u_t, z_t)} w_{t-1}^{(i)}, \quad (3.65)$$

where in probability $p(z_t|x_{1:t}^{(i)}, z_{1:t-1}, x_0)$ in equation (3.63), poses of the i^{th} particle from time 1 to $t-1$, $x_{1:t-1}^{(i)}$, and observations from time 1 to $t-1$, $z_{1:t-1}$ are replaced with the map constructed by them, m_{t-1} .

Similar to particle filtering, explained in Section 3.2.9, by choosing the motion model as the proposal distribution, the weight-update becomes

$$w_t^{(i)} \propto p(z_t|x_t^{(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}. \quad (3.66)$$

In equation (3.66), the likelihood of the measurement is conditioned on the pose and the map developed so far. This is an important relation which uses not only the current pose but also the available map, to update the weight of each particle. This relation is used in the next chapter to develop an algorithm for multiple-robot SLAM.

3.3.4.4 DP-SLAM

The distributed particle (DP) filter, proposed by Eliazar *et al.* [87], is a fast and reliable implementation of particle filtering for SLAM. DP-SLAM is developed for grid mapping using laser rangers. One of the problems which are identified in SLAM is the need for a high number of particles. Assume that each particle holds a map and the pose. If a particle has pose error, then its map will have an accumulated error. Therefore to have a better accuracy, more particles are required for SLAM compared with localization, where the map is known. Since maps require more memory to be maintained, the algorithm becomes demanding. FastSLAM solves this problem by keeping track of the map uncertainty in a simple Gaussian form. However, DP-SLAM solves this problem by proposing an efficient data structure for maintaining grid maps, which is called distributed particle mapping. To further elaborate the problem, assume there are p particles each holding a map of size m . After a restamping, to copy maps an operation is required which is of the order $\mathcal{O}(mp)$. In the DP-SLAM

solution, particles are related through their ancestry. If particle p is sampled from particle q , then q is a parent particle and p is a child particle. If two particles, p_1 and p_2 , are derived from the same particle, they are sibling particles. By introducing this concept, it is possible to keep track of changes in maps, rather than copying all maps. However the challenge with this structure is that if maps are required for localization, then all particles should be revisited to find the root ancestry and recover the map. This operation can be time consuming. To solve this problem and keep the ancestry tree, two actions are taken. First, any particle with no child is pruned away to its first root particle. Therefore, only ancestors of the current generation of particles are present in the ancestry tree. Second, if a particle has only one child, that branch of the tree is removed. To do this, that branch is collapsed. This merging makes updating faster. The resulting ancestry tree with P particles is a minimal tree with three properties.

- The tree has P leaves,
- The tree has a branching factor of at least 2,
- The depth of the tree is no more than P .

To minimize the required memory to keep maps, a data structure is proposed. The main idea behind it is to associate particles with maps, rather than associating maps with particles. Therefore, there is only one occupancy grid map in which particles are associated with different cells of the map. Each grid stores a balanced tree.

3.3.5 Smoothing Solutions for SLAM

Smoothing solutions solve the full SLAM problem, in which the whole trajectory of the robot is estimated. In this section, two general smoothing methods are reviewed: GraphSLAM and sub-map matching.

3.3.5.1 GraphSLAM

Generally, in GraphSLAM, poses of a robot are represented as nodes in a graph. The edges connecting nodes are modelled with motion and observation constraints. These constraints are extracted and calculated by the SLAM front-end. Next, these constraints need to be optimized to calculate the spatial distribution of the nodes and their uncertainties [3]. This is performed by the SLAM back-end [88]. Usually the optimization by the back-end takes more time and memory than extracting the constraints. Therefore it runs at a lower frequency.

Calculating constraints by observations is a data association problem. The data association usually operates locally; therefore, the search space is limited by considering the uncertainty of the pose estimates.

GraphSLAM was first formulated by Lu and Milios [17]. They optimized a system of equations to decrease the error caused by constraints. Since then, researchers have proposed different solutions for GraphSLAM including the work by Gutmann *et al.* [16], relaxation on a mesh by Howard *et al.* [89], multilevel relaxation by Frese *et al.* [90], iterative alignment by Olson *et al.* [41], square root simultaneous location and mapping (SAM) by Dellaert *et al.* [59], incremental smoothing and mapping (iSAM) by Kaess [91] and finally three works by Grisetti *et al.* in [92, 93], and hierarchical optimization for pose graphs on manifolds (HOGMAN) [94].

In HOGMAN, the optimization is performed by Gauss-Newton with sparse Cholesky factorization. The state space is represented as a manifold and therefore avoids the parameterization singularities for estimating 3D rotations. This optimization is performed using a hierarchical method. The lower level of the hierarchy represents the original constraints, while the higher level corresponds to the abstract structural information. When a new constraint is added to the the problem, first the highest level is reoptimized completely. Then at the lower level, only the regions of the map which are changed are updated. This results in a globally consistent map with reduced

computational requirements

The front-end of HOGMAN is an extension of Olson's work [95] from 2D to 3D. A new node is added to the graph, whenever the robot travels a minimum distance. The edges between nodes are created by scan matching. A key responsibility of the front-end is to detect loop closure. To do this an approximation of the conditional covariances of all nodes given the current node is calculated. If a node falls under 3σ of the current node, scan matching is performed between these two nodes. If the result is successful, an edge is created between the two nodes.

The task for the back-end is to find the configuration of the nodes in a way that maximizes the likelihood of the observations. Put mathematically, for two nodes, i and j , let x_i and x_j be two poses and z_{ij} and Ω_{ij} be their mean and information matrix of the edge that connects node i to node j . Furthermore, assume e_{ij} is the innovation of the observation (*i.e.*, the difference of the expected observation and the sensed observation). GraphSLAM seeks to find a configuration of nodes, x^* , that minimizes the negative log likelihood of all measurements

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{(i,j) \in C} e_{ij}^T \Omega_{ij} e_{ij}, \quad (3.67)$$

where C is the set of pairs of indices that an observation constraint exists. Linearizing equation (3.67) by Taylor expansion, the following relation is the result:

$$H \Delta x^* = b, \quad (3.68)$$

where the matrix H is a sparse information matrix of the system. This allows equation (3.68) to be solved efficiently by sparse Cholesky factorization. Once Δx^* is solved, it is added to the initial estimate to find the optimized poses, (*i.e.*, $x = \hat{x} + \Delta x^*$) In all solutions of GraphSLAM, the main challenge is how to solve equations (3.67) and 3.68. HOGMAN uses iterative Gauss-Newton on a manifold to solve for equa-

tion (3.68). In the iterative approach, each time the result is used as an initial estimate for the next iteration. The main difference between HOGMAN and other methods is that HOGMAN linearizes on a manifold rather than in Euclidean space. Linearization in Euclidean space is not a valid assumption for SLAM, since orientation angles span non-Euclidean space [94, 96].

iSAM [91] is another graphSLAM solution which is based on the original batch square root SAM [59] by F. Dellaert. iSAM uses the same principles used in HOGMAN but it provides an incremental solution (full trajectory, with no filtering) by updating a factorization of the information matrix. At any time, exact marginal covariances are available for loop closure and data association problems.

3.3.5.2 Sub-map Matching

Sub-map matching is also related to GraphSLAM. Sub-map matching is an efficient method of mapping where small local maps are matched with each other and mosaiced to produce a global map. This prevents global error accumulation due to odometry and measurement inaccuracies, and therefore is a good solution for large scale and outdoor environments. Bailey’s work [12], hierarchical SLAM by Estrada *et al.* [97], the Atlas framework [15] and Tectonic SAM [98] are examples of such a solution. In the Atlas framework [15], a global map management solution is proposed which can close loops in structured environments. Atlas provides view-based local maps using an EKF and attaches them together by comparing the normals of the maps. In Bailey’s work [12] a similar solution is proposed which is developed using a 2D feature-based SLAM. A similar approach is proposed in Tectonic SAM [98] where optimization is performed to align local maps. Local maps in this method are generated using square root information smoothing [59].

3.3.6 Bio-inspired Solutions

Artificial intelligence has also been used to solve the SLAM problem. In these solutions, filtering or smoothing are realized using fuzzy logic or neural networks. For instance, in [99] a solution based on fuzzy logic is proposed to tune covariance values of the measurement model. These values are important for the stability of the filter. The ratSLAM [100] is a technique that models a rodent's brain using neural networks. In fact this method is a neural network-based data fusion using a camera and an odometer.

The presented solutions for SLAM are summarized in Table 3.3.

3.3.7 2D and 3D SLAM

Robotic environments are in general 3D; therefore, robots translate in three directions and rotate around three axes. Assume a robot moves in an environment where the coordinates are based on the right-handed coordinates system. In this system, X and Y axes define a horizontal plane. The X axis points towards left, the Y axis points away from you, and the Z axis points down. Rotations around X , Y , and Z axes are referred to as roll, pitch, and yaw angles, respectively. In most cases, especially indoors, the robot has no change in the Z direction and has negligible roll and pitch angles. In these cases, the SLAM problem can be simplified to estimating only x and y coordinates and the yaw angle; this is referred to as 2D SLAM. However, in outdoor environments and when the robot demonstrates changes in the Z direction or it has considerable rotation for roll and pitch angles, then SLAM requires 3D pose estimation.

The complexity of 3D SLAM is not as simple as estimating three more parameters. This is mainly because most sensory information lacks a full 3D perception and therefore it becomes complicated to estimate parameters which are not observable directly. Moreover, if there exist sensors providing full 3D information, the complexity of pro-

Table 3.3: Comparison of some common SLAM techniques.

SLAM Method	Description
EKF-SLAM [83]	<ul style="list-style-type: none"> • based on the extended Kalman filter <p>Pros: works well if features are distinct Cons: adding a new feature to state space needs quadratic time, requires feature extraction in feature-based SLAM, cannot identify absence of a feature</p>
EIF-SLAM [84, 101]	<ul style="list-style-type: none"> • based on the extended information filter <p>Pros: measurement updates are performed in constant time, effective for multiple-robot SLAM due to additivity of information Cons: information matrix needs to be sparsified. Recovering map and the pose requires large matrix inversion</p>
PF-SLAM [10]	<ul style="list-style-type: none"> • based on particle filtering. FastSLAM is an efficient implementation of particle filtering <p>Pros: effective for loop closure, performs full and online SLAM, logarithmic complexity in number of features, no need for parametrization Cons: quality of the estimation is dependent on the number of particles</p>
DP-SLAM [87]	<ul style="list-style-type: none"> • a fast implementation of particle filtering for SLAM <p>Pros: effective for large scale maps, optimizes memory requirements Cons: requires processing to recover maps</p>
GraphSLAM [17, 91, 94]	<ul style="list-style-type: none"> • uses smoothing techniques to estimate the trajectory and the map <p>Pros: the whole trajectory is updated Cons: computationally demanding, hard to recover covariances from the information form</p>
Sub-map Matching [76, 98]	<ul style="list-style-type: none"> • matches small local maps to make a large global map <p>Pros: the whole trajectory is updated, suitable for large scale environments Cons: size of local maps should be adjusted</p>
Bio-inspired SLAM [99, 100]	<ul style="list-style-type: none"> • based on artificial intelligence <p>Pros: efficient, as it mimics animals' brains, no mathematical model is required Cons: error-prone, requires training or parameter tuning, training can be time-consuming</p>

cessing algorithms increases significantly. For instance, in the simplest form, solving a nonlinear equation for n variable, requires an $n \times n$ matrix inversion, which by the fastest implementation, is of the order $\mathcal{O}(n^{2.373})$ [102].

Stereo cameras, such as Kinect camera, and 3D laser rangers, such as Velodyne laser scanner, are among the well-known sensors used for 3D SLAM. In [103] and [104], a method called rgbdSLAM is proposed to perform 3D SLAM using stereo and Kinect cameras. This method will be explained in Chapter 7 in detail. In [105], Segal *et al.* propose a 3D scan matching for Velodyne 3D laser ranger. This method will also be explained in Chapter 7.

3D modelling of the environment has a lot of applications like better utilizing the environment by architectures, integrating models with 3D game platforms, and helping rescuers to identify the environment clearly. Polygon-based SLAM or multi-planar methods are 3D mapping methods in which the world is represented as multiple planes mosaiced together [79, 82]. This method is mainly used in indoor environments. It is also possible to perform 3D SLAM or 3D mapping using two perpendicular scanning laser rangers. This method is introduced in Chapter 7 in more detail.

3.4 Background on Multiple-robot SLAM

This section provides the background to multi-agent systems and current solutions for multiple-robot SLAM.

3.4.1 Multi-agent Systems

In a multi-agent system, there are a few key issues which need to be taken into account. The way these issues are handled makes each solution unique. These issues are phrased in the form of the following four questions. First, what type of data is shared among agents? Second, how is this data shared among agents? Third,

where is this data processed? Finally, how is the processing performed? In the next few subsections, each of these questions are answered, respectively, in the context of multiple-robot simultaneous localization and mapping.

3.4.1.1 Data Sharing

Sharing data among robots is a fundamental issue in multiple-robot SLAM. Past approaches to collaborative SLAM can generally be categorized based on whether they share raw sensor data [28] or processed data [2]. Raw sensor data means that sensed information such as laser ranger measurements and wheel odometry readings are not processed. In processed data, such as a map or poses of robots, sensor readings are processed through filtering or smoothing or other methods. Sharing raw sensor data results in more flexibility but requires high bandwidth and reliable communication between robots as well as more processing power. In contrast, sharing maps uses less bandwidth and there is no need to process raw data redundantly; however, the performance is dependent on the quality of maps. The latter method is usually referred to as map merging or map fusion. The work by Thrun [106] or Howard's work [28] are examples of sharing raw laser and odometry data among robots. Works by Carpin *et al.* [30] and Birk *et al.* [2] are examples of sharing processed maps between robots. The choice of the method depends on several factors such as the application and the available resources. In this work, a combination of both approaches is used.

3.4.1.2 Data Communication

Availability of a system to share data among robots is a key requirement in multiple-robot SLAM. Information between robots can be exchanged via communication channels which might not be available at all times, in all places. Bandwidth and coverage of the communication network are two important factors in the performance of SLAM. Using contemporary available communication technologies, wireless networks with

high bandwidth can easily be set up and utilized. Let us assume three robots in a collaborative work want to map an unknown environment. Suppose each is equipped with a scanning laser ranger and two encoders. Assume the laser ranger operates at 40 Hz and each scan has 1000 beams, where each beam is represented by a float number of the type float64. A number of the type float64 is composed of 8 bytes or 64 bits. Therefore, in total the required bandwidth will be $1000 \times 40 \times 64 = 2.56$ Mb/s. By assuming that there are two encoders operating at the same rate using float64 numbers, the required bandwidth for encoders will be $2 \times 40 \times 64 = 5.12$ Kb/s which is very small and negligible in comparison. Therefore, each robot's wireless interface needs a throughput of $3 \times 2.56 = 7.68$ Mb/s. A low cost wireless interface can handle up to 100 Mb/s; therefore, even more robots can be added to the team. A simple solution to deal with coverage problem and communication blackouts is to buffer data on each robot and process it later. In this work it is assumed that a communication network is available for all robots to share their data; however, in extremely unreliable networks, it is possible to extend the proposed work to be robust against frequent networking issues using algorithms such as [33] and [35].

3.4.1.3 Data Distribution

In multiple-robot SLAM, data can be processed in a centralized manner, where all data is sent to a central agent or a central work-station. The central agent processes the incoming data and provides required information and feedback to other agents. The work in [107] is an example of this solution. The other approach is the decentralized solution, where each robot receives data from others and processes the data to make its own perception of the environment. Obviously this requires that robots have enough computational power to respond to processing demands. The method proposed in [28] is an example of this solution. Hybrid solutions are also applicable, where some robots process data and some use the results only. The work in [29] is an example of such a

solution where 80 robots were cooperating at different levels to make a global map. In this work, a decentralized approach is used, where each robot shares raw sensor data with other robots and develops its own perception.

3.4.1.4 Data Processing

The choice of the method for estimating the poses of robots and the global map depends on many factors such as the available memory, processing capability and type of the sensory information. Generally, the spectrum of SLAM algorithms includes many different approaches each with remarkable capabilities. At the two extreme ends of the spectrum lie two methods: EKF-SLAM and GraphSLAM [3]. EKF-SLAM approximates the nonlinearity of the system by linearizing the system model. Also, EKF-SLAM is computationally expensive; each time acquired information is resolved into a probability distribution which makes it proactive but slow. GraphSLAM solutions such as iSAM [91] and HOGMAN [88] also depend on approximation by Taylor expansion. Its difference with EKF-SLAM is that it accumulates information and therefore is considered to be a lazy and an off-line algorithm [3]. The particle filtering approaches are nonlinear filtering solutions; therefore, the system models are not linearized. Although particle filtering is computationally expensive, it is shown that a good implementation such as FastSLAM [3] and grid mapping [43] can reduce the complexity of the algorithm.

In this work, particle filtering is selected for estimating the joint posterior of poses and the map; however, developing a multiple-robot SLAM algorithm requires addressing its complexity issues. By proper selection of the proposal distribution for multiple-robot SLAM and applying Rao-Blackwellization, the number of the particles is reduced and hence space complexity is improved compared with straightforward implementation of particle filtering.

3.4.2 Current Solutions for Multiple-robot SLAM

In this section filtering, smoothing, and other solutions in the literature for multiple-robot SLAM are reviewed.

3.4.2.1 EKF-SLAM

Zhou *et al.* [21] propose robot rendezvous to solve the unknown initial pose problem. In robot rendezvous, robots should meet either by random or by arrangement, at least once. This solution is feature-based and uses EKF for filtering robot poses and landmark positions. The advantage of this method is that there is no need to have common features in the maps of the robots; however, the robots need to see and detect each other at a point. When robots are in the line-of-sight, robot-to-robot measurements are used to calculate the transformation between maps. Then maps are merged using the calculated transformation. Once the maps are merged, it is highly possible that there are duplicates of the same landmark due to the uncertainty of the position of the landmark. The so called *Sequential Nearest Neighbour Test* is proposed to detect these cases and combine duplicate landmarks. The test is based on the Mahalanobis distance [108]. If the distance of two landmarks in the fused map is smaller than a threshold, they are considered to be duplicates. To update the map, state and covariance row and column of the duplicate landmarks are deleted. If the landmarks' error is bigger than the distance between them, then this method will not be effective. Therefore, to avoid false duplicate detection, this process is performed in the vicinity of two robots, where it is more likely to have less error in the position of the landmarks. This process is performed sequentially. The experiments show the consistency of the final fused map.

3.4.2.2 EIF-SLAM

Inherently, the EIF is more suitable for a multi-robot system than EKF. This is because information has an additivity property in EIF. Similar to EIF, SEIF is a suitable approach for multi-robot mapping because of its additivity. Thrun *et al.* [20] propose a multi-robot feature-based SLAM. Their approach uses the additivity of the SEIF to develop a complete solution for multi-robot SLAM. By the additivity property, robots can integrate their information by simply adding them. Multi-robot SEIF highlights the importance of finding the relative poses at the first encounter between robots.

3.4.2.3 PF-SLAM

Thrun proposes a probabilistic multiple-robot view-based SLAM in [106]. This method combines maximum likelihood mapping with Monte-Carlo localization. In this method, the probabilistic motion model and the probabilistic measurement model based on scan matching are used. This method is robust and can close loops. However, there are two main limitations. First, the robots must begin their mapping in nearby locations to have overlaps in their range scans (to have large overlaps in their initial maps). Second, the initial poses of the robots are assumed to be known approximately prior to the start of the mission.

Howard [28] extends the work in [106] and proposes a multiple-robot SLAM using Rao-Blackwellised particle filter. In this method, it is assumed that either robots' relative poses are given or robots will meet each other at a point. At the meeting point, which happens within the line-of-sight, robots identify their relative positions using cameras and unique fiducials mounted on them. At the first meeting point a particle filter is applied to the data in reverse temporal sequence to fuse all the data from the initial point. The future meetings are ignored. This approach is effective and the maps and poses are updated in real-time, but it cannot be applied when robots cannot see

each other or they do not know their relative poses initially. Also, no uncertainty for relative poses is considered. When the robots have mutual observations, mutual laser observation which are induced by other robots are excluded. This approach has been tested on a team of four robots.

In [29], Howard *et al.* extend the work in [28] to include path planning and exploration with a central agent and a team of 80 robots.

Carlone *et al.* [109] also applies a particle filter for multiple-robot SLAM as in [28]. Their contribution of the work is including the uncertainty of the relative transformation in the mapping. The main drawback of the work is that it does not hold a joint posterior for trajectories and relative poses of robots. Also, the robots exchange data only in encounters. If encounters does not happen frequently, it can cause buffer overflow or it can occupy all processing resources at encounters for processing all buffered data. This work also suffers from exponential space complexity.

3.4.2.4 Map Merging

Map merging or map fusion [2] is a solution to the multiple-robot SLAM, in which map data from robots are fused to generate a global map. Let a transformation be composed of a 2×2 rotation matrix R_ψ and a 2×1 translation vector T as follows:

$$R_\psi = \begin{bmatrix} \cos \psi & -\sin \psi \\ \sin \psi & \cos \psi \end{bmatrix}, T = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix}, \quad (3.69)$$

Let m_1 and m_2 be two occupancy grid maps. Assuming that m_2 is merged into m_1 , the map merging problem is defined as: find a rotation matrix, R_ψ and a translation vector, T which transforms m_2 such that the overlaps of m_1 and the transformed m_2 , m'_2 , fall squarely on top of each other. To do this it is required to maximize a

verification index defined on the merged map:

$$(R_\psi, T) = \underset{\psi, \delta_x, \delta_y}{\operatorname{argmax}} V(m_1, m'_2), \quad (3.70)$$

where m'_2 means that m_2 is transformed according to R_ψ and T . $V(\cdot)$ is a criterion that evaluates the merging process and will be explained in the following sections in more detail.

This optimization is not easy to solve analytically and most methods [2, 30] use an exhaustive search to find a transformation matrix and then check equation (3.70) to see if it is satisfied. Moreover, to avoid false results in highly repetitive structures, it is very important to identify overlaps robustly.

Fig. 3.10 depicts an example of map merging with three maps. The overlap between map_1 and map_2 is shown with a dashed ellipse and the overlap between map_1 and map_3 is shown by a solid line ellipse. To find the required transformation, the first step is to identify overlaps, then use the overlaps to calculate the alignment and finally verify the results using equation (3.70).

The map merging problem can be interpreted as a special case of image registration in computer vision where images are maps with special geometry. However, the problem is that in multiple-robot map merging, the percentage of overlap between maps is usually low.

3.4.2.5 GraphSLAM

GraphSLAM (see Section 3.3.5) has also been used for multiple-robots, for instance, C-SAM by Andersen *et al.* [110], decentralized SLAM by Cunningham and Dellaert [111], and the work proposed in [107]. C-SAM focuses on feature-based map merging and is a batch algorithm. In [107], Kim *et al.* extend iSAM for multiple robots based on multiple pose graphs in which the relationship between multiple pose

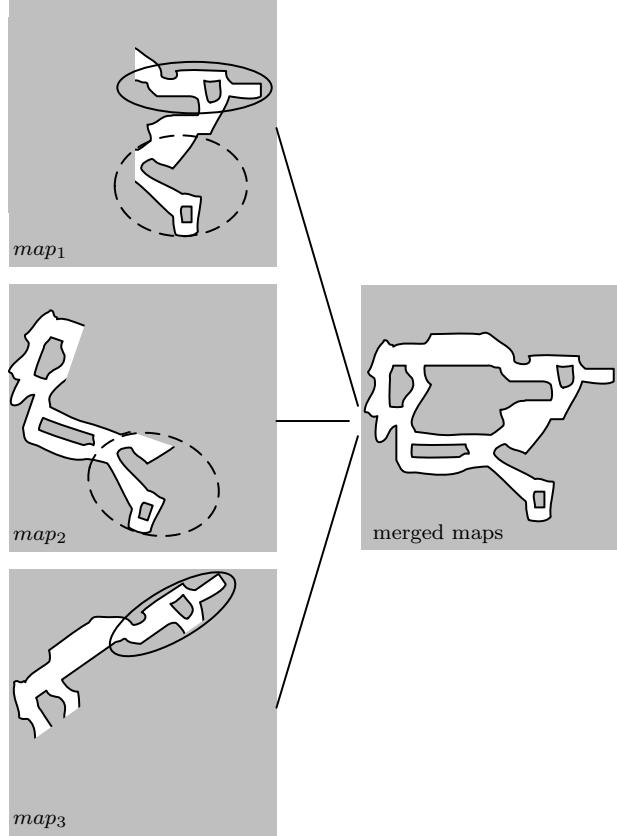


Figure 3.10: An example of map merging. Three maps are merged to form a complete map of the environment. The overlap between *map₁* and *map₂* is shown with a dashed ellipse. The overlap between *map₁* and *map₃* is shown with a solid line ellipse.

graphs are formulated and optimized to generate a consistent solution. This solution considers constraints between robots, when they see each other (direct encounter), and constraints due to observing the same environment (indirect encounter). The proposed method is based on the concept of the *base node*, introduced in the Tectonic SAM. base nodes are nodes in the pose graph which depict initial pose of robots and are initially set to be the origin. If encounters happen between robots, base nodes are updated to the actual initial poses. To extend iSAM for multiple robots, direct and indirect encounters are considered as constraints in the optimization process. Therefore, the same solution and factorization which is used for single-robot SLAM is easily applied to the multiple-robot SLAM.

3.4.2.6 Constrained Local Sub-map Filter

Williams *et al.* [112] propose a sub-map based solution for multiple-robot SLAM, called constrained local sub-map filter (CLSF). CLSF reduces the complexity of multiple-robot SLAM by periodically fusing the local sub-map of the features, generated from features nearby the robot, to the global map. This way, every observation is not fused directly into the global map, and therefore the complexity decreases. Moreover, since the uncertainty of the pose and features in the local map is usually small, the data association problem is also improved. In addition, data association can even be deferred until an improved local map is generated.

3.4.2.7 Manifold Representation

Howard *et al.* [113] propose a multiple-robot SLAM in 2D environments using manifold representation. In this solution, the trajectory of the robot forms a spiral in the 3D space. The main advantage of the manifold representation is solving the ambiguity in incremental mapping. This approach deals with temporary inconsistency of maps and also extends it to the multiple-robot SLAM. In this solution, multiple-robot SLAM is performed in three main steps: incremental localization and mapping, loop closure, and island merging. Incremental localization and mapping is the process of exploring the unknown environment while incrementally making a map as a manifold. While performing incremental mapping, loop closure happens when two largely separated parts of the map are brought together by manifolds. Island merging happens when two unconnected parts of the manifold are fused together into a single representation. Island merging in multiple-robot SLAM is used when robots with unknown initial poses see each other. This process fuses local maps of the robots into a global map. This algorithm handles mutual observations and loop closure; however, because of the complexity of the algorithm it poorly scales with the size of the team.

3.5 Summary

In this chapter, the background to multiple-robot SLAM was introduced. First, sensing devices and their applications were discussed. Then, probabilistic estimation and filtering solutions were investigated. Then, current solutions for single-robot SLAM were briefly reviewed. Finally, multi-agent systems and current solutions for multiple-robot SLAM were reviewed.

Chapter 4

Map Merging

This chapter presents four proposed solutions for multiple-robot SLAM. These solutions solve the map merging problem and are proposed for view-based SLAM.

4.1 System Overview

As mentioned in the previous chapters, map merging is an effective solution for multiple-robot SLAM in which maps of individual robots, whether feature-based or view-based, are fused together to produce a global map. In this context, the individual map of each robot is also referred to as a local map. In this chapter, the state of a robot includes 2D Cartesian coordinates and the orientation of the robot.

4.1.1 Problem Statement

Map merging has benefits such as the need for less bandwidth and being robust against occasional communication failures; however, there are some problems which will be listed and explained as follows.

4.1.1.1 Relative Poses of Robots

To fuse local maps from robots and make a global map, transformations between maps are needed. These transformations are related to the initial poses of robots which are generally unknown. To find these transformations, one solution is to use the local maps where overlaps between the maps need to be identified and used for finding the transformation between the maps.

4.1.1.2 Uncertainty of the Relative Poses

Once the relative transformation between any two maps is calculated, maps can be aligned and merged. However, the transformation is not deterministic and has a level of uncertainty which is represented by a covariance matrix. To transform maps and align them, this uncertainty should be taken into account. In other words, once a map is transformed, it should reflect the uncertainty of the transformation as well.

4.1.1.3 Updating Maps

Two aligned maps can be fused to generate a global map. This needs a procedure to use the information from corresponding cells from both maps, *i.e.*, the probabilities from corresponding cells should be combined to reflect the effect of probabilities from both maps.

4.1.1.4 Complexity

The required time and memory to perform the map fusion is an important factor in the efficiency of the process. Modern robots do not suffer from memory limitations; however, the map fusion should run in real-time.

4.1.2 Problems with Existing Methods and Motivations

There are few works on map merging with view-based SLAM and occupancy grid maps. The main works include adaptive random walk (ARW) map merging [2] by Birk *et al.* (also used in [25]) and the works by Carpin *et al.* [30, 114]. The proposed methods in [2] and [30] are relatively slow, while the approach in [114] fails to fuse maps with fewer overlaps as will be illustrated later. Also none of them takes into account the uncertainty of the relative poses and they do not mention how the maps are updated. To solve these issues, different solutions have been proposed. Experimental results and discussions show the effectiveness of the proposed solutions. More detailed information on problems and motivations is provided when each solution is presented in the next sections.

4.1.3 Description of the Proposed Methods for Map Merging

In this section each of the four solutions is briefly reviewed with an explanation of which problem each was to solve.

4.1.3.1 Map Segmentation

This solution aims to find the relative transformation matrices of maps. In this solution, edges of maps are processed to extract features which are used to find the transformation. Results are tuned to provide better alignments.

4.1.3.2 Neural Networks Approach

This novel solution finds the relative transformation matrices between the maps while reducing the complexity of the operation. In this method maps of the environment are learned through competitive self-organizing maps (SOM). During the learning procedure, maps are clustered. These clusters, which represent salient information of the maps, are used to find the relative pose of robots. Processing clusters reduces

the complexity of the algorithm significantly. There is no known similar research or application.

4.1.3.3 Probabilistic Generalized Voronoi Diagram

In this innovative solution, a generalized Voronoi diagram (GVD) is extended to reflect the probabilistic nature of the occupancy grid maps. This extension, which is called the probabilistic generalized Voronoi diagram (PGVD), is used to find overlaps and the transformation between the maps. The probabilistic Voronoi diagram is used to match parts of the maps which have less uncertainty. This results in a better and more robust alignment. Moreover, experiments show that this is a fast solution compared with others. Also, in this solution uncertainty of the transformation matrix is taken into account and a new method is proposed to fuse aligned maps.

4.1.3.4 Hough Peak Matching

In this new method, maps are taken into the Hough space and all processing are preformed in the Hough space. It is shown that it is easier to find overlaps and the transformation matrix in the Hough space. This solution is also compared with other methods to demonstrate its efficiency and applicability in different environments.

4.1.4 Advantages and Disadvantages

The set of solutions proposed for finding the transformation matrix and map merging can handle different types of environment. For instance, the Hough peak matching is suitable for environments with many straight walls while the map segmentation is more suitable for featured environments.

The PGVD approach has the advantage that it preferentially matches more certain parts of the maps which results in a more robust solution. Moreover, the uncertainty of the transformation has also been taken into account.

The neural networks approach is the first ever method which uses artificial intelligence to solve the map merging problem. It is shown that the proposed solutions work faster than current solutions.

More detailed information on advantages and disadvantages of each solution is presented later.

4.1.5 Summary

To summarize this section, four solutions for map merging were presented briefly. Each of these solutions will be presented in more detail in the following four sections of the chapter. Each section highlights which problems of multiple-robot SLAM are targeted.

4.2 Map Segmentation

This section presents a solution for the problem of map merging. The proposed solution to merge maps is based on matching segments in the maps that are not straight lines. Featured segments carry more information than straight lines making the matching process more robust since unique features are less likely to be incorrectly matched. This novel map feature extraction is based on processing histograms of map segments. In a typical grid map which is full of similar right angle corners and straight walls, it can robustly identify unique map features and use them for map fusion.

4.2.1 Problems with Existing Methods and Motivations

Map merging, as an effective solution for multiple-robot SLAM, requires finding the transformation between the robots' maps. As explained in the previous section, current solutions are not fast enough; therefore, robots are not able to move quickly enough through the environment and tasks cannot be achieved promptly.

The proposed method is motivated by the fact that developing a fast method to find the transformation between the maps of the robots is very important in performance of robotic missions. In this work, a multi-step algorithm is proposed whereby maps are processed to extract important information for merging the maps.

4.2.2 Background: Radon Transform

In this section, the Radon transform [115] is briefly introduced. The Radon transform is the projection of the image intensity along a radial line oriented at a specific angle. The Radon image is generated by computing a Radon transform and varying the Radon angle, θ , of the radial line onto which the original image is being projected. For a given 2D map, $m(x, y)$, the Radon transform along the radial line of angle θ is defined as:

$$r_\theta(x') = \int_{-\infty}^{\infty} m(x' \cos \theta - y' \sin \theta, x' \sin \theta + y' \cos \theta) dy', \quad (4.1)$$

where

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \quad (4.2)$$

The Radon image generated by computing a Radon transform is a collection of all possible projections for a larger set of angles. In a Radon image, the horizontal axis represents all possible angles and the vertical axis is the Radon transform for individual angles.

4.2.3 Description of the Method

The proposed map fusion algorithm in this research is based on a search and verification algorithm. Fig. 4.1 shows the algorithm for two robots with unknown relative positions and each with its own local map, $m_1 = map_1$ and $m_2 = map_2$ respectively.

This algorithm is scalable and can be used for more than two robots. Maps in this

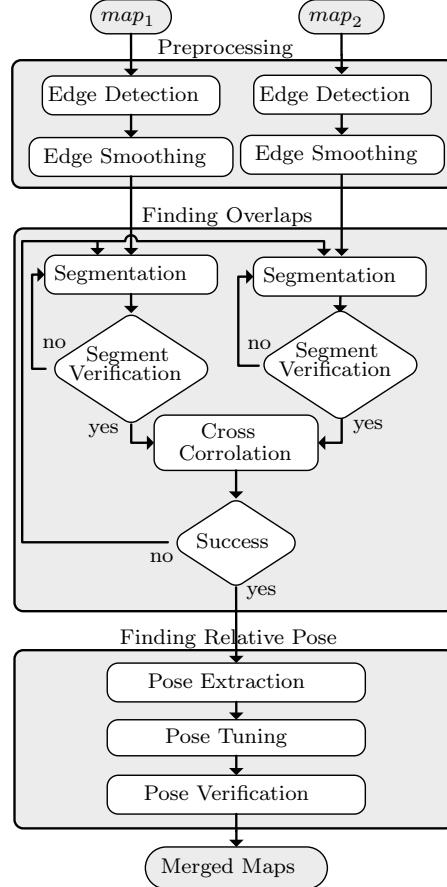


Figure 4.1: The proposed map fusion algorithm. Two input maps, map_1 and map_2 , are fused by finding their relative transformation matrix. No prior information is available regarding the relative position of two respective robots.

algorithm are assumed to be in the form of the occupancy grid maps [3].

According to Fig. 4.1, the diagram of the system, the inputs are map_1 and map_2 .

The algorithm is composed of three main steps:

- **Preprocessing:** In this step maps are processed to reduce the computational demand in the next steps and obstacles are represented in an abstract form.

- **Finding overlaps:** Common parts between maps are extracted in this step.

These overlaps are used in the next step.

- **Finding relative pose:** In this step, overlaps are used to calculate the transformation matrix. The results are tuned and verified in this step.

First, in the *Edge Detection* block, the Canny edges of the maps are extracted [66]. Then, in the *Edge Smoothing* block, the extracted edges are smoothed to facilitate processing in the subsequent blocks. In the *Segmentation* block, a segment of obstacles in the smoothed map is selected. The segment is passed through the *Segment Verification* block to ensure that it contains enough unique geometric information to be used for comparison. The segmentation and verification process continues until an acceptable segment is found or a maximum number of search iterations is reached. The same process is done on the other map. If acceptable segments from both maps are found then the selected segments are tested for similarity in the *Cross Correlation* block. If the selected segments are deemed to be congruent, then they are passed to the *Pose Extraction* block where a rough estimate of the relative pose of the robots is determined. In the *Pose Tuning* block, the orientation of the approximate relative pose is further tuned using the Radon transform [115] and then the translation elements of the relative pose are tuned by a similarity index. In the *Pose Verification* block, the final results are verified. If pose verification rejects the calculated transformation, then the process can start over with other segments generated in the *Segmentation* block or at a later time when maps have more overlaps. This is not shown in the block diagram to keep it simple. In the following sections, each block is explained in detail.

4.2.3.1 Edge Detection

The first processing block is *Edge Detection*. Here, noisy measurements are removed from both maps using a smoothing filter and then edges are detected using the Canny edge detection method [116]. Canny edge detection involves the following steps: 1) The input image is filtered by convolving it with a Gaussian filter, 2) The gradient

of the image, J , is calculated and has components along the horizontal and vertical axes as given by:

$$J = J_x \hat{i} + J_y \hat{j}. \quad (4.3)$$

3) From the derivatives, the edge strength and direction can be determined by:

$$E_s = \sqrt{J_x^2 + J_y^2}, \quad (4.4)$$

$$E_o = \arctan\left(\frac{J_y}{J_x}\right). \quad (4.5)$$

4) Non-maximum suppression is performed on the map. In non-maximum suppression, cells whose values of E_s are not larger than their neighbours along the direction perpendicular to the edge orientation in E_o are set to zero. 5) A hysteresis thresholding is performed: given a high threshold t_h and a low threshold t_l ($t_h \geq t_l$) cells are marked as edges if either: a) $E_s \geq t_h$ or b) $E_s \geq t_l$ and connected to an established edge point \hat{e} by other points with strength $E_s \geq t_l$ in the direction of the edge at \hat{e} . Once this five step process is complete, a binary image is generated where a ‘1’ represents an edge cell.

This process is necessary because the occupancy grid mapping shows boundaries of obstacles as thicker than they are in reality due to the noise in sensor measurements. Using edge detection, it is possible to extract the exact obstacle boundaries to be used in the other blocks so that the map can be processed more accurately and efficiently.

4.2.3.2 Edge Smoothing

The next block is *Edge Smoothing*. Here, the Canny edges are used to estimate the most probable locations of the obstacles. The objective is to remove redundant edges caused by the noisy laser sensor measurements and select the most likely locations of the obstacles.

The error of laser measurements follows a Gaussian distribution. As an example,

Fig. 4.2-a depicts a 2D Gaussian distribution along a curve where the distribution at each point is given by:

$$f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}}, \quad (4.6)$$

where σ_x and σ_y are variance values and μ_x and μ_y are mean values for two uncorrelated variables, x and y .

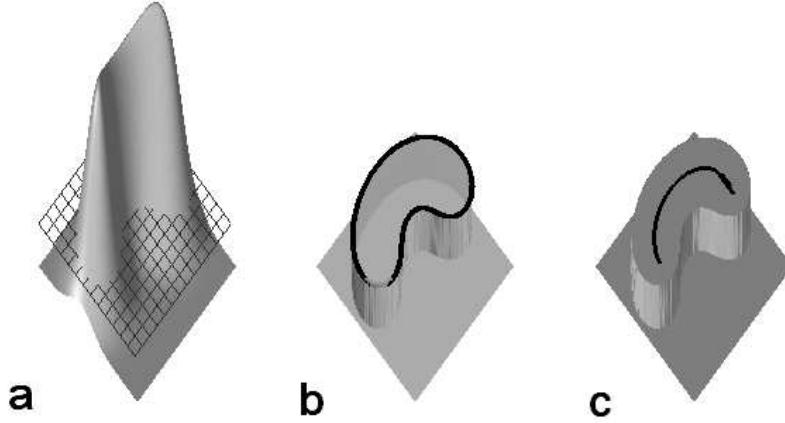


Figure 4.2: **a)** A 2D Gaussian distribution with no truncation. The meshed plane is the truncation surface. **b)** Any point below the surface is set to zero to produce a truncated Gaussian distribution. **c)** Middle point of edges.

Through the edge detection operation, the Gaussian distribution from Fig. 4.2-a is truncated based on the Canny threshold t_h and t_l to produce the edges shown in Fig. 4.2-b. However, the edge detection algorithm has produced two edges where only one obstacle exists in reality. These two edges are reduced to one, which resides at the peak of the Gaussian distribution as shown in Fig. 4.2-c.

In the implementation, it is therefore reasonable to assume that pairs of nearby edges actually represent only a single edge, which can be assumed to be at a location equidistant from the two original (redundant) edges. The location of the new edge is found by performing averaging in the horizontal and vertical directions as described in detail in Algorithm 4.2.1.

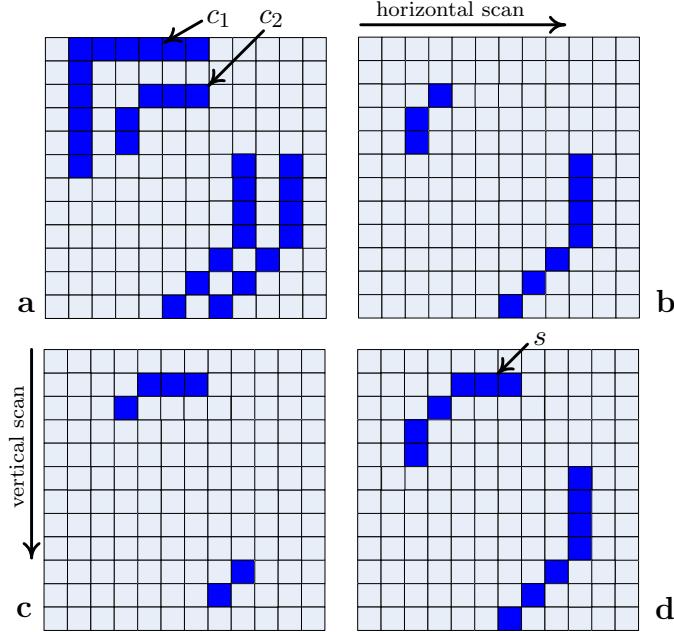


Figure 4.3: Vertical and horizontal scans for a simple edge map ($d_t = 6$). **a)** Original map: two adjacent edges, c_1 and c_2 are shown in the map. **b)** Output after the horizontal scan. **c)** Output after the vertical scan. **d)** Output after both scans. Final s curve for c_1 and c_2 , mentioned in equation (4.7), is the output of the process.

More specifically, suppose that we have two adjacent edges c_1 and c_2 that are output from the Canny edge detector. Assuming that the two edges are close enough that they actually only represent one true obstacle, the most probable position of the obstacle, s , is given by

$$s = c_1 \oplus c_2, \quad (4.7)$$

where the operator \oplus performs middle curve extraction on two dimensional curves of c_1 and c_2 by doing an averaging in the horizontal and vertical directions, and outputs the set of points s that represent the best approximation of the actual location of the obstacle.

It is difficult to generate closed mathematical representations of c_1 and c_2 , so the operator \oplus can be defined as operating over the detected edges by scanning in the vertical and horizontal directions. Algorithm 4.2.1 explains this process for a vertical scan. The input to the algorithm is the edge map and the output is a smoothed edge

Algorithm 4.2.1 Scanning an edge map in a vertical direction.

Input: Edge map: $E^{r \times c}$.

Output: Smoothed edge map: $S^{r \times c}$.

```
1:  $S \leftarrow 0$ 
2: for  $i = 1 \rightarrow r$  do
3:    $n \leftarrow 0$ ,  $cell \leftarrow \emptyset$ 
4:   for  $j = 1 \rightarrow c$  do
5:     if  $E(i, j) = 1$  then
6:        $n \leftarrow n + 1$ 
7:        $cell(n) \leftarrow j$ 
8:     if  $n = 2$  then
9:       if  $1 < cell(n) - cell(n - 1) < d_t$  then
10:         $k \leftarrow \text{round}((cell(n) + cell(n - 1)) / 2)$ 
11:         $S(i, k) = 1$ 
12:     end if
13:      $n \leftarrow 0$ ,  $cell \leftarrow \emptyset$ 
14:   end if
15:   end if
16: end for
17: end for
```

map. First the output matrix is initialized to a zero matrix (line 2). In a vertical scan, (lines 2-17), all columns are processed for each row. Occupied cells in each row are identified in line 5 and stored in a temporary variable (line 7). Once two occupied cells are identified (line 8), their distances are processed. If the distance between two occupied cells is larger than one and less than some specified threshold d_t , (line 9) then the middle point of the two occupied cells is considered to be the most probable occupied cell (line 10) and this cell is marked as an occupied cell in the output map (line 11). This process continues for all cells in each row. A similar process is performed in the horizontal direction. Fig. 4.3 shows a simple example of the scanning process ($d_t = 6$). The major benefit of smoothing edges is that the complexity and scale of the resulting maps are reduced without any significant loss of data.

One major advantage of this approach is that the location of resulting edges should be invariant to the quality of the sensor used, since noise is being filtered out. This

is important for map merging in the subsequent steps since if robots have variable sensing capabilities, they should still be able to merge their maps.

4.2.3.3 Segmentation

The next block is the *Segmentation* block. The objective of this block is try to select unique areas of each map so that they can be compared in the hope of finding parts of the maps that are shared. These shared features are used to find the relative transformation matrix between the two robots. The Segmentation block looks for segments of the map which have unique geometric properties, similar to the way the human brain functions [62]. As a human being, our brain tries to find shared parts in different images and then identifies the transformation based on those shared parts. Specifically, parts of the map that contain curves or angles are selected as good candidate segments. It is also noted that segments should be made from the occupied cells. Because a normal occupancy grid map will have significantly less occupied cells than free or unknown cells.

The segment selection process begins with choosing a random start point in the map, called p_0 , which can be any point that represents an occupied cell. p_0 is a reference for other points. A fixed number of points, n , along the edge are then identified and represented as:

$$p_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}, i = 0..n, \quad (4.8)$$

where p_i is the i^{th} point of the segment and x_i and y_i denote the discrete points.

The segment is then processed to determine if it contains unique geometrical characteristics by generating and analyzing two other vectors: the Euclidean distance vector, d and the differential angle vector, δ :

$$d = \begin{bmatrix} d_1 & d_2 & \dots & d_n \end{bmatrix},$$

$$\delta = \begin{bmatrix} \delta_1 & \delta_2 & \dots & \delta_{n-1} \end{bmatrix}, \quad (4.9)$$

where

$$d_i = \|p_i - p_0\|,$$

$$\delta_i = \angle(p_i - p_0) - \angle(p_{i-1} - p_0). \quad (4.10)$$

The vector d of the segment is invariant with respect to any rotation or translation of the segment. The vector δ will differentiate between two segments which are mirror images.

The two characteristic vectors are a unique representation, λ , of the original segment P as given by:

$$\lambda = (d, \delta). \quad (4.11)$$

Theorem 4.2.1. *From λ and the starting location of the segment, p_0 , it is possible to reconstruct the entire segment P uniquely.*

Proof. By induction:

Base case

p_0 must be known.

Induction step

Assume that k points in the segment are known. From d_{k+1} , it is known that the point p_{k+1} lies on a circular locus of radius d_{k+1} centered at p_0 . Consider that points p_0 and p_{i-1} are already located. Let $\phi_k = \angle(p_k - p_0)$, which is the known angle of point k given that its exact location is already known. Let $\phi_{k+1} = \angle(p_{k+1} - p_0)$, which is unknown. From equation (4.10), $\delta_{k+1} = \phi_{k+1} - \phi_k$. Rearranging gives $\phi_{k+1} = \delta_{k+1} + \phi_k$. Since both terms on the right hand side are known, the angle ϕ_{k+1} can be calculated. From the angle that p_{k+1} makes with p_0 , the point on the circular locus can be uniquely identified. \square

4.2.3.4 Segment Verification

To identify whether a segment is a good candidate for map fusion, a histogram method is used. Two histograms are constructed to verify a segment. The first one is the distance histogram, hd which is generated from the distance vector, d , and the second is the arc histogram, ha , which is generated from the arc vector which is defined as:

$$\omega = \begin{bmatrix} \omega_1 & \omega_2 & \dots & \omega_{n-1} \end{bmatrix}. \quad (4.12)$$

Each arc is defined as

$$\omega_i = d_{i+1} \sum_{j=1}^i \delta_j, \quad (4.13)$$

where $i = 1, 2, \dots, n - 1$.

The histograms are generated using a tunable number of bins, where the values in the bins correspond to the number of elements of the vector that fall between the boundaries specified for that bin. The number of bins should be chosen to sufficiently represent the information in the segments. Algorithm 4.2.2 explains this process for a $1 \times n$ vector, $v^{1 \times n}$ with a $1 \times m$ bin vector specified by $b^{1 \times m}$. The output of the algorithm is the histogram depicted by a vector $h^{1 \times m}$. The number of bins chosen is important in order to get acceptable result. In this research, it is determined experimentally over a set of trials.

Algorithm 4.2.2 Generating a histogram for a vector.

Input: $v^{1 \times n}, b^{1 \times m}$.

Output: $h^{1 \times m}$.

```

1:  $h_i \leftarrow 0, \forall h_i, i = 1, \dots, m$ 
2: for  $i = 1 \rightarrow n$  do
3:   for  $j = 1 \rightarrow m$  do
4:     if  $(v_i \geq b_j) \& (v_i < b_{j+1})$  then
5:        $h_j \leftarrow h_j + 1$ 
6:     end if
7:   end for
8: end for
```

The distance histogram, hd , and arc histogram, ha are represented as:

$$hd = \begin{bmatrix} hd_1 & hd_2 & \dots & hd_{nd} \end{bmatrix},$$

$$ha = \begin{bmatrix} ha_1 & ha_2 & \dots & ha_{na} \end{bmatrix}, \quad (4.14)$$

where nd is the number of the bins and hd_j , $j = 1, 2, \dots, nd$ is the value corresponding to each bin for hd , and na is the number of the bins and ha_k , $k = 1, 2, \dots, na$ is the value corresponding to each bin for ha .

If the segment is a wall or flat surface, then the distance histogram will have a flat shape, and the arc histogram will have almost all bins empty except for one. If there is an angle in the segment, then the distance histogram will not be flat and more than one bin of the arc histogram will be non-empty.

To verify the acceptability of a segment, the following conditions should be met:

- 1) $hd_j \neq 0, \forall j = 1 \dots nd$
- 2) The distance histogram should not be smooth:

$$\max(hd) - \min(hd) \geq h_d, \quad (4.15)$$

where h_d is a predefined threshold value.

- 3) The segment should not be discontinuous. A discontinuity will cause a large value in the arc vector, so the condition:

$$\max(|\omega_i|) \leq \omega_c, \forall i = 1 \dots n - 1, \quad (4.16)$$

where ω_c is a predefined threshold value that will reject segments with discontinuities. This novel histogram method is fast and is successful at identifying segments with unique geometric properties.

It is important to note that the size of the occupancy grid map cells is fixed across

all robots in the team. The result is that matching segments will be the same size across all maps.

4.2.3.5 Cross Correlation

In the *Cross Correlation* block, the distance and arc histograms of the two selected segments from the two maps are compared using a cross correlation technique. The histograms are normalized and then a 2D cross correlation operation is applied. If the distance and arc histograms are similar, then the result of each cross correlation will be close to one and the segments are deemed a match. Otherwise, another segment from m_2 is selected and the process continues. If, for the selected segment from m_1 , no correspondence from m_2 can be found, then another segment from m_1 should be selected. If no shared segment can be found after a certain number of trials, then the two maps cannot be fused. To increase the confidence level of the result of the histogram matching, a correlation of the distance and arc vectors is also applied. Given two vectors, $u^{1 \times n}$ and $v^{1 \times n}$ the following relation is used to calculate the correlation coefficient as a measure of similarity.

$$r = \frac{\sum_{i=1}^n (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{(\sum_{i=1}^n (u_i - \bar{u})^2)(\sum_{i=1}^n (v_i - \bar{v})^2)}}, \quad (4.17)$$

where \bar{u} and \bar{v} are mean values of vectors u and v respectively.

4.2.3.6 Pose Extraction

After finding matching segments from both maps, the *Pose Extraction* block is executed. First, the approximate relative rotation is determined, and then the rotation is used to find the approximate translation. To determine the rotation, a line is fitted for each segment using minimum least-squares error. The difference in the slopes of the two fitted lines gives the approximate rotation angle between the two maps. If

the slope of the fitted line for the selected segment of m_1 is a_1 and of m_2 is a_2 , then the approximate rotation angle, α_{app} is:

$$\alpha_{app} = \arctan2(a_1) - \arctan2(a_2) \quad (4.18)$$

where $\arctan2$ is the arc tangent function that bounds the output to the interval $(-\pi, \pi]$. Once the approximate rotation has been found, the approximate translation can be determined. First, the approximate rotation is applied to align both segments. Next, the approximate translation is computed to be the difference in the centroids of the two segments as given by:

$$tr_{app} = \begin{bmatrix} x_{app} \\ y_{app} \end{bmatrix} = \sum_{k=1}^{n_1} \frac{R_{\alpha_{app}} P_k^{seg1}}{n_1} - \sum_{k=1}^{n_2} \frac{P_k^{seg2}}{n_2}, \quad (4.19)$$

where $R_{\alpha_{app}}$ is the rotation matrix based on α_{app} , P^{seg1} and P^{seg2} are the sets of points in segments 1 and 2 respectively. n_1 and n_2 are the cardinalities of P^{seg1} and P^{seg2} respectively.

4.2.3.7 Pose Tuning

After finding the approximate rotation and translations, these elements need to be tuned. The tuned pose will be extracted in two steps using the *Radon Transform* and a *Similarity Index*. First, to find the tuned rotation a Radon transform is used [115]. One of the characteristics of the Radon image is that the peak points occur when the Radon angle, θ , aligns with straight line segments that occur in the image. The approximate rotation is used to ensure that the correct peak is selected from the Radon transform.

As a result, it is possible to resolve the exact rotation by looking for peaks in the

Radon images of both maps that are close to the approximate angle already computed.

$$\alpha_{ext} = \alpha_{app} + \delta_{tuning}, \quad (4.20)$$

$$\delta_{tuning} = \operatorname{argmin}_{\theta} (\mathcal{R}_{\theta=0:180}(m_1)) - \operatorname{argmin}_{\theta} (\mathcal{R}_{\theta=0:180}(T_{x_{app}, y_{app}, \alpha_{app}}(m_2))), \quad (4.21)$$

where α_{ext} is the exact or tuned rotation angle and $\mathcal{R}_{\theta=0:180}(map)$ is the Radon image of the input map over the specified domain of angles (θ). $T_{x, y, \theta}(map)$ means that the input map is translated according to the values of x, y and rotated by θ .

After tuning the rotation, the translation is tuned using a *similarity index*. A performance index is used which has been introduced in [2]. This index measures the similarity of two maps over some desired region. When there is more overlap between two maps, there will be an extremum in the index. This index is composed of two components:

$$agr(m_1, m_2) = \#\{p = [x, y]^T | m_1(p) = m_2(p)\}, \quad (4.22)$$

$$dis(m_1, m_2) = \#\{p = [x, y]^T | m_1(p) \neq m_2(p)\}, \quad (4.23)$$

where the operator $\#$ over a given set returns the cardinality of the set. The similarity index is defined as:

$$J_{sim}(m_1, m_2) = dis(m_1, m_2) - agr(m_1, m_2). \quad (4.24)$$

The function $agr(\cdot)$, the agreement index, is the number of known cells with equal status in both maps (either both occupied or both free). The function $dis(\cdot)$, the disagreement index, is the number of cells which are known in at least one map and have unequal status. A smaller disagreement index and larger agreement index will result in a larger similarity index, J_{sim} . The maximum absolute value represents the best translational match between the two maps. To find the maximal value, an

exhaustive search in the neighbourhood of the approximate translation is done. It is challenging to use guided search algorithms like genetic algorithms or particle swarm optimization because in many cases the best solution may occur very close to very poor solutions. The size of the search space is determined by experience based on the performance of the approximate translation method. In addition, if the total allowable time for searching can be determined, the size of the search space can be specified such that the search is completed within the allotted time. Usually a search space with a radius of ten cells around the current approximate solution is acceptable considering the required time constraints and accuracy of the previous approximate result. The best candidate translation vector is selected by determining the one with the best similarity index:

$$tr_{ext} = \begin{bmatrix} x_{ext} \\ y_{ext} \end{bmatrix} = \underset{\mathcal{S}}{\operatorname{argmax}}(|J_{sim}(m_1, T_{x,y,\alpha_{ext}}(m_2))|), \quad (4.25)$$

$$\mathcal{S} = \{(x, y) \in \mathbb{N}^2 | x_{app} - \delta_x < x < x_{app} + \delta_x, y_{app} - \delta_y < y < y_{app} + \delta_y\} \quad (4.26)$$

where tr_{ext} is the tuned translation vector and \mathcal{S} is the search space, defined as a rectangle centered at (x_{app}, y_{app}) with dimensions $2\delta_x \times 2\delta_y$.

4.2.3.8 Pose Verification

After finding the best candidate for the transformation matrix, a verification is performed in the *Pose Verification* block. In case of false matching, a verification is required to ensure the results are accurate. The verification process acts as a check on whether the previous blocks operated correctly. If the percentage of the verification is lower than a threshold, the results are rejected and the process can start over with other segments or at a later time when the maps have more overlaps. The

verification method, called similarity verification, uses the ratio of $agr(\cdot)$ and $dis(\cdot)$ as given by:

$$V(m_1, m'_2) = \frac{agr(m_1, m'_2) \times 100\%}{agr(m_1, m'_2) + dis(m_1, m'_2)}, \quad (4.27)$$

$$m'_2 = T_{x_{ext}, y_{ext}, \alpha_{ext}}(m_2) \quad (4.28)$$

where m_1 and m_2 are two input maps and $V(m_1, m'_2)$ is the verification index. m'_2 is translated according to x_{ext} and y_{ext} and rotated based on α_{ext} . If $V(m_1, m'_2)$ is more than a threshold, then the proposed transformation can be accepted. The maximum verification percentage occurs when the maps are perfectly aligned.

4.2.4 Advantages and Disadvantages

The proposed solution analyzes the maps of the robots and finds overlaps by looking for common map segments. The edge smoothing algorithm reduces the computational demand, but it might introduce uncertainty to the map. Searching for segments takes the majority of the processing time and this can be a limitation on the size of the map. However, the experimental results, which compare processing time of the proposed algorithm with adaptive random walk map merging [2], show its efficiency.

4.2.5 Contribution

In the mapping process, uncertainty in system modelling, measurement and processing causes noise accumulation in the grids of the map. For example, a single occupied point (cell) in the real world can cause many surrounding cells to appear occupied because of the sensor and process uncertainty. As a result, more processing power will be required to handle the falsely detected occupied cells. As a contribution of this work, to solve this problem, edges of obstacles are extracted by the Canny edge detector algorithm, then, since the artificial thickness of the obstacles caused by sen-

sor noise will result in multiple edges being produced, a novel edge smoothing method is applied to remove redundant edges.

To find overlaps between the smoothed edge maps, usually an exhaustive search method is required over the whole domain of the maps. As another contribution of this work, map features are extracted and processed to reduce the domain of the search. Most map feature extraction methods like the Split-Merge, the Radon and the Hough transform algorithms rely only on straight lines and ignore featured segments. In this work a novel map feature extraction method is introduced that analyzes the histogram of occupied map segments. If the histogram of a segment satisfies specific criteria, it is accepted as a featured segment of the map.

The next step is to determine the transformation using the extracted features. First an approximate transformation is determined by matching features, then the results of the transformation are tuned to provide a better transformation.

The result is a multi-step robust and accurate map merging algorithm that is capable of merging maps with minimal overlap from multiple robots without knowing their relative poses.

4.2.6 Experiment

Two experiments are performed on real-world indoor environments with two CoroBot robots. The first experiment is performed in a simple environment using two robots and the second one is done in a more complex environment with three robots.

4.2.6.1 Case1: Two CoroBot Robots

This test is performed by two CoroBot robots in an indoor environment. All details of the algorithm are explained during the description of this experiment.

CoroBot is a differential-wheeled robot built by CoroWare Inc. It is equipped with two High Speed Phidget Encoders, a 3DM-GX1 Microstrain IMU, and a Hokuyo



Figure 4.4: CoroBot Robots, equipped with laser rangers and encoders.

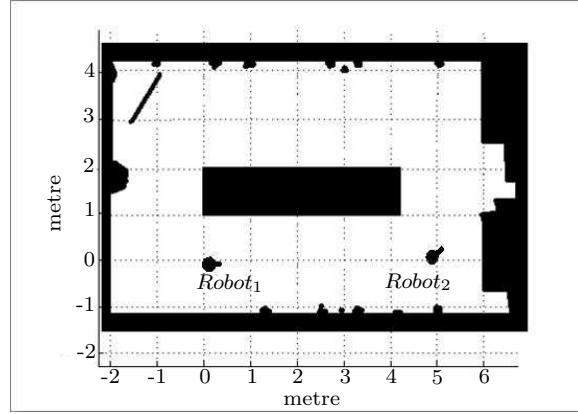


Figure 4.5: A simple test environment with two robots for experiment 1.

UBG-05LN laser ranger, as shown in Fig. 4.4. The laser ranger sensor has a range of 4500 mm with an angle resolution of 0.36° resulting in 513 points for each scan.

Fig. 4.5 shows the approximate floor plan and dimensions of an indoor environment used for experimentation. This room has been surveyed by two robots from different initial positions which are assumed unknown.

Data from onboard sensors are fused with a cascaded EKF. Specifically, data from the laser rangefinder and IMU are filtered sequentially to provide an estimate of the pose of the robot.

Fig. 4.6-a shows the local map provided by the first robot, and Fig. 4.7-a shows the local map provided by the second robot. In both figures, grey cells are unknown, white cells are free, and black cells are obstacles. The size of all maps is 400×400

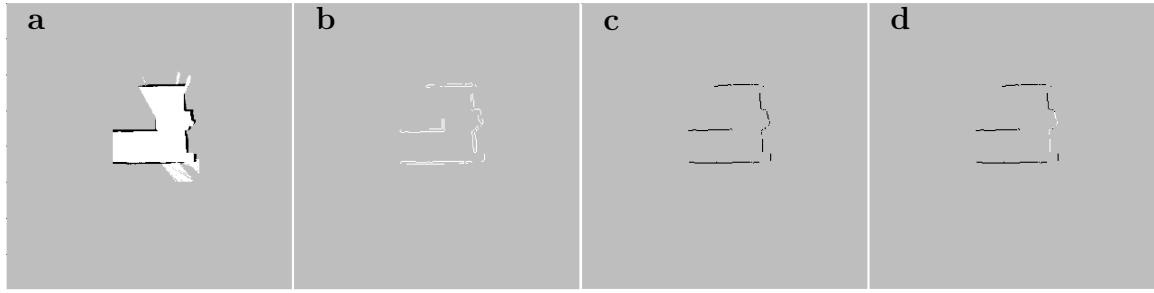


Figure 4.6: **a)** m_1 , occupancy grid map of the test environment provided by the first robot. **b)** Canny edges **c)** Smoothed edges **d)** Matching segment in white colour

cells with identical resolution.

Fig. 4.6-b and Fig. 4.7-b show the effects of the *Edge Detection* block on the two maps. These figures show boundaries of occupied cells which will be used in the next block.

Fig. 4.6-c and Fig. 4.7-c show the results of the *Edge Smoothing* block for m_1 and m_2 , respectively. Extracted edges are smoothed to remove redundant edges caused by sensor noise.

Fig. 4.6-d and Fig. 4.7-d show two matching segments from m_1 and m_2 in white which are the results of the *Segmentation*, *Segment Verification* and *Cross Correlation* blocks.

The criterion to select these segments for matching is implemented in the *Segment Verification* and *Cross Correlation* blocks. In the first block, arc and distance histograms are used to verify a segment. As explained, a segment with a flat distance

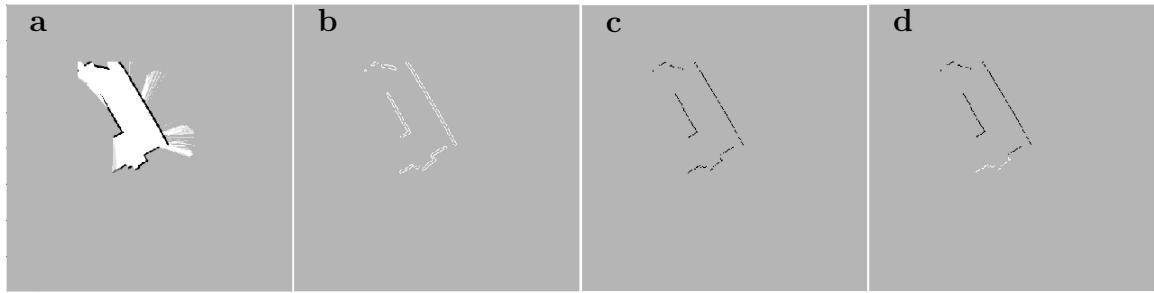


Figure 4.7: **a)** m_2 , occupancy grid map of the test environment provided by the second robot **b)** Canny edges **c)** Smoothed edges **d)** Matching segment in white colour

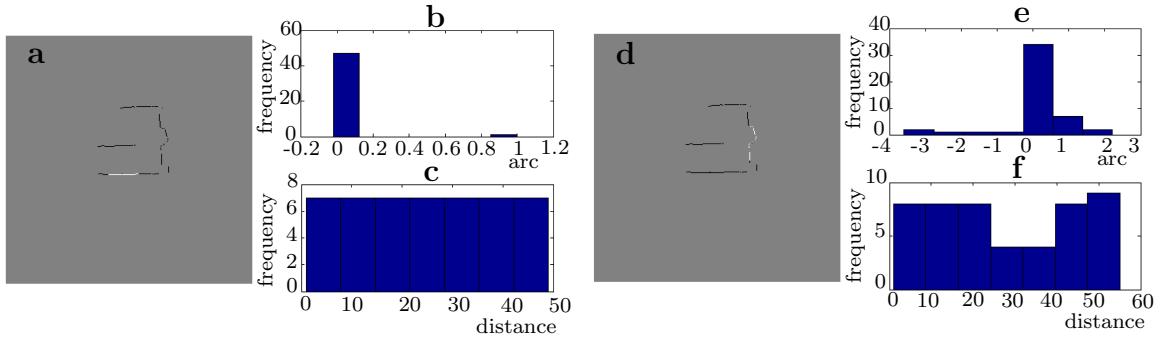


Figure 4.8: **a)** A straight wall segment **b)** Arc histogram of the straight segment **c)** Distance histogram of the straight segment, **d)** A featured wall segment **e)** Arc histogram of the featured segment **f)** Distance histogram of the featured segment

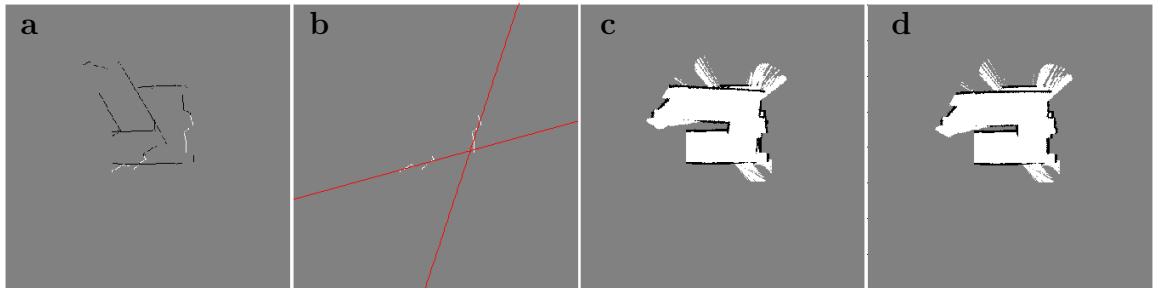


Figure 4.9: **a)** Both maps in the same coordinates with marked segments, **b)** Fitted lines on segments to extract relative orientation, **c)**, Both maps in the coordinates of m_1 after approximate transformation, **d)** Rotation improvement after Radon transformation

histogram or a sharp arc histogram is not a good segment for map merging. Fig. 4.8-a shows a flat segment with its distance and arc histograms in Fig. 4.8-b and Fig. 4.8-c, respectively. Fig. 4.8-d shows a non-flat segment. Its distance and arc histograms are depicted in Fig. 4.8-e and Fig. 4.8-f, respectively. The flat segment is not a good candidate for map merging while the second segment is. If segments have acceptable histograms, they are matched by the cross correlation operation and if the similarity is high enough, they are used for map merging.

Fig. 4.9-a shows both maps on the coordinates of m_1 with identified common segments. Fig. 4.9-b shows the fitted lines of the two similar segments. The angle between these two lines is an approximation of the relative orientation of the two maps. After finding the approximate rotation, equation (4.19) is used to find the

approximate translation. These operations are performed in the *Approximate Pose Extraction* block.

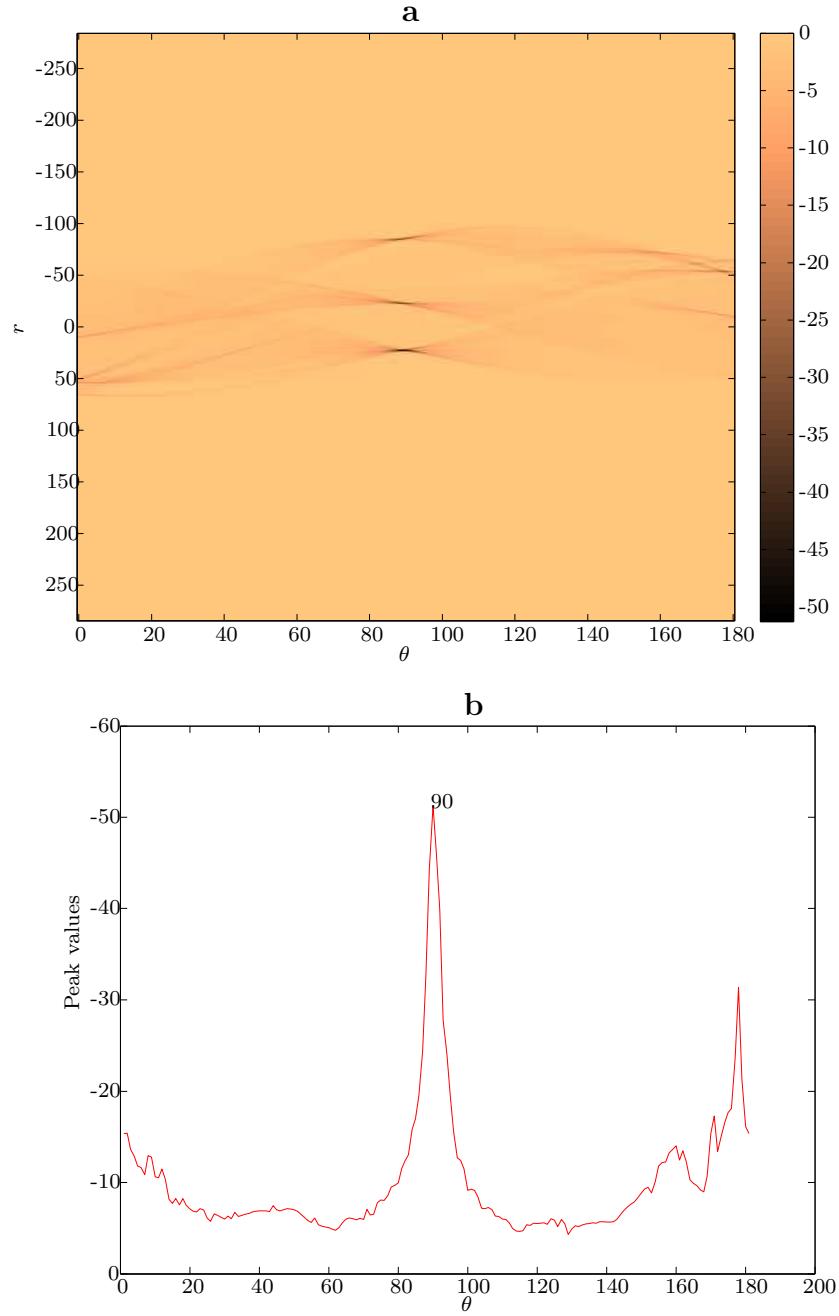


Figure 4.10: **a)** Radon image of m_1 over 180 degrees **b)** peak point of the Radon image

Fig. 4.9-c shows both maps in the coordinates of m_1 after the approximate transformation. It is clear from the figure that this transformation is not accurate and needs

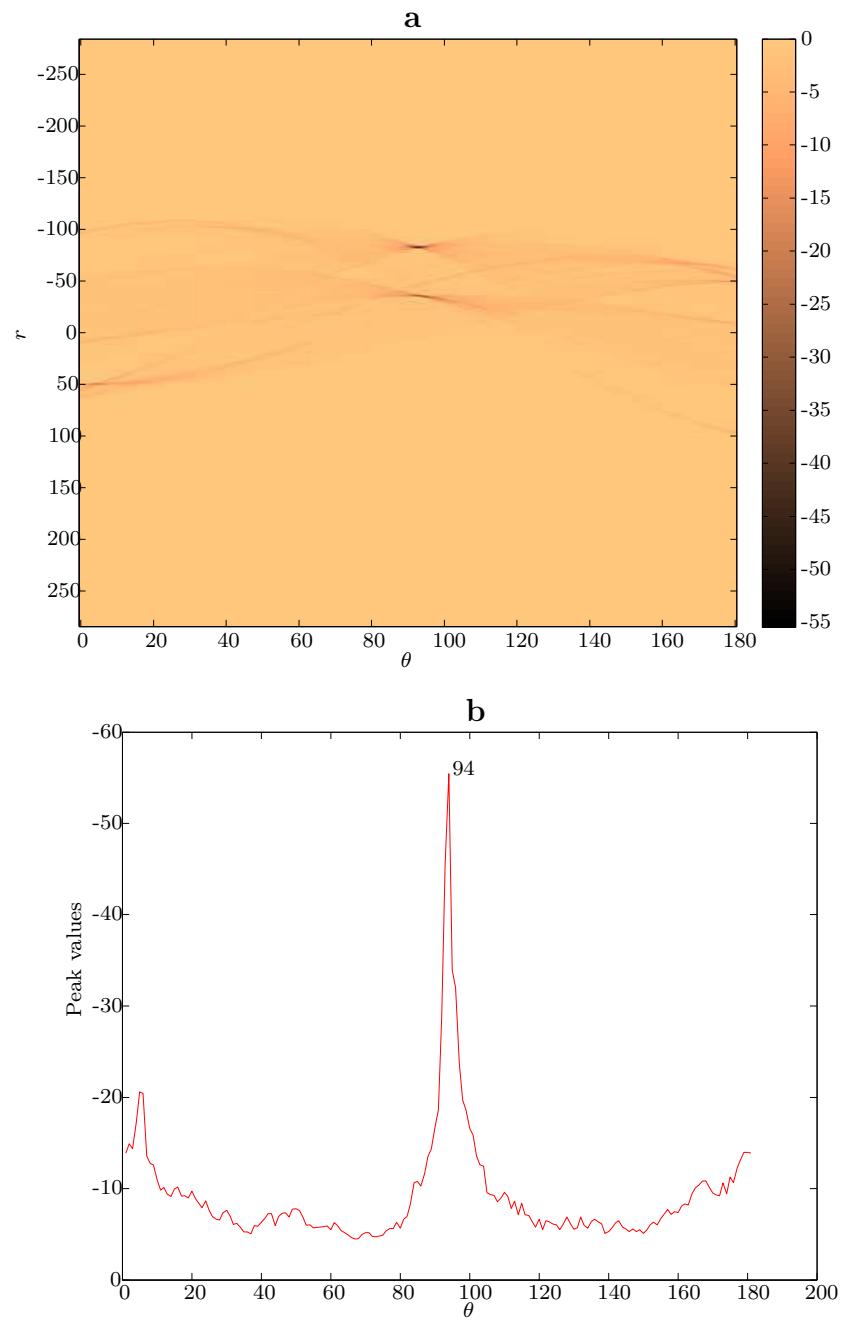


Figure 4.11: **a)** Radon image of m_1 over 180 degrees **b)** peak point of the Radon image

further modification.

The next step is tuning the rotation angle and the translation which are performed in the *Pose Tuning* block. First the orientation is tuned by the Radon transform. Fig. 4.10-a and Fig. 4.11-a show the Radon images for m_1 and m_2 respectively after applying the approximate transformation. As Fig. 4.10-b and Fig. 4.11-b show, the peak point of the Radon image for m_1 happens to be at 90° while for the approximately transformed m_2 it is 94° . The difference of 4° is the tuning angle that should be added to the original approximate angle as determined by equation (4.20).

Fig. 4.9-d shows both maps in the coordinates of m_1 after applying the proposed Radon tuning method. Clearly, this figure shows the effective improvement over the relative rotation angle shown in Fig 4.9-c.

The next step is tuning the translation. Fig. 4.12 shows a 3D representation of the similarity indices over a set of translation vectors along the vertical and horizontal axes following a raster scan pattern. As indicated, the best similarity index occurs at the best translation, which is shown by a circle.

Finally, Fig. 4.13 shows the final overlapped maps with exact rotation and translation elements.

To verify the results, the proposed method in the *Pose Verification* block is used. The verification index using the similarity index introduced in equation (4.27) is 95% which is the peak of all verification indices over the search space. This is shown in Fig. 4.14.

4.2.6.2 Case2: Three CoroBot Robots

To demonstrate the effectiveness of the proposed methods, an experiment is performed using three robots in an environment with the approximate size of 17×10 metres. Fig. 4.15 shows the approximate floor plan of the test environment.

Each robot generates a local occupancy grid map of the environment. At certain

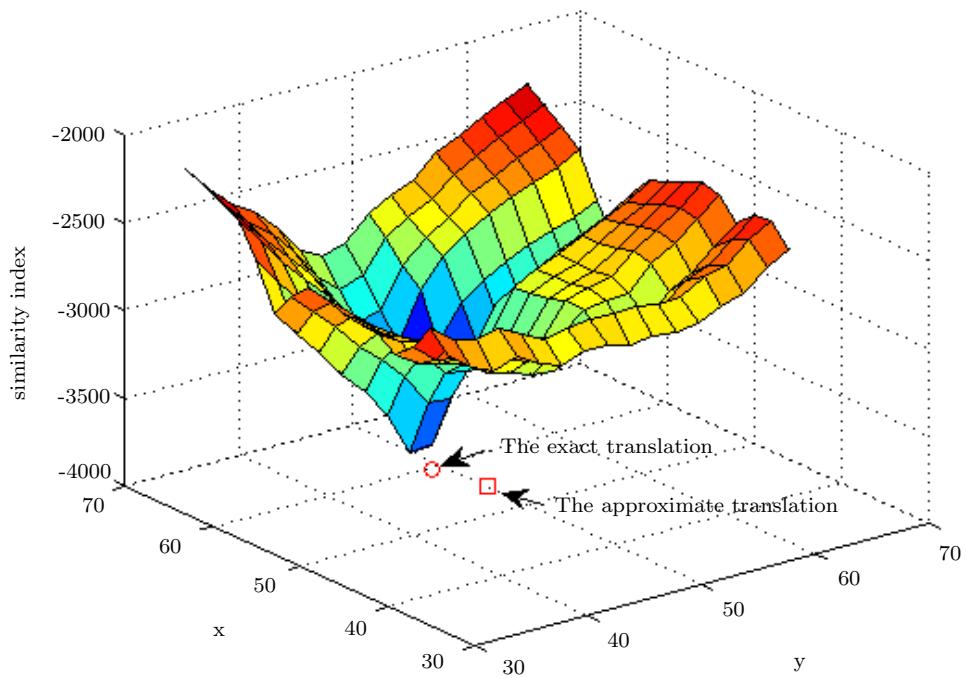


Figure 4.12: 3D representation of the similarity index over the search space. The best similarity index occurs at the best transformation.

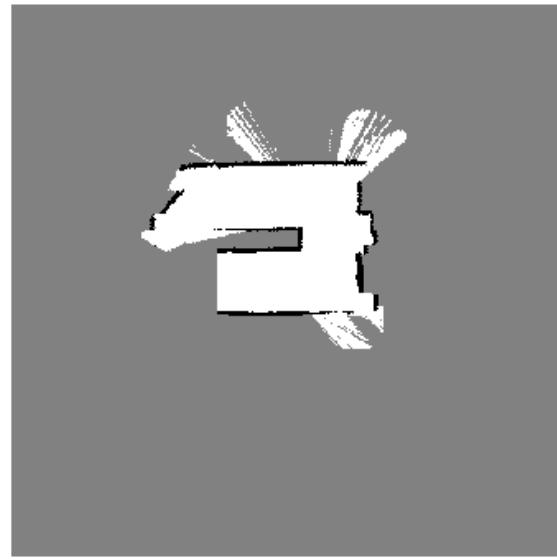


Figure 4.13: Final alignment of both maps in the coordinates of m_1

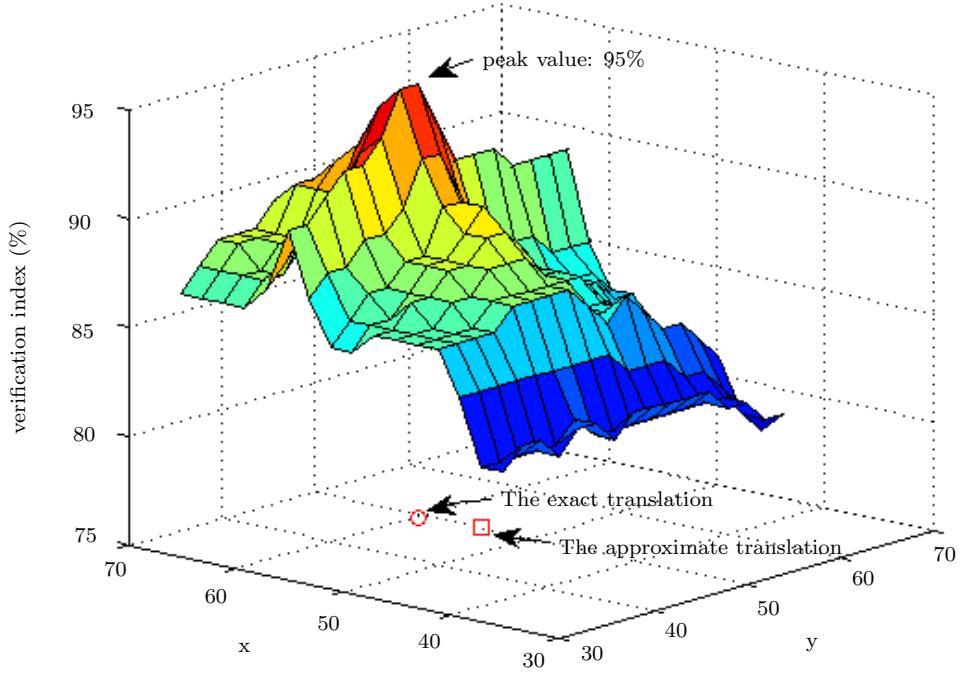


Figure 4.14: 3D representation of the verification index over the search space for the fused maps. The maximum value occurs at the best translation.

specified time intervals, the robots share their local maps with each other. Fig. 4.16 shows the three local maps generated by the three robots. Note that the left part of the test environment was out of the range of the laser rangefinders for all three robots. The three maps, m_1 , m_2 , and m_3 , are provided by $robot_1$, $robot_2$, and $robot_3$, respectively. The proposed map fusion algorithm is assumed to be done on $robot_1$ with m_2 and m_3 being integrated into m_1 .

Fig. 4.17-a shows the result of map fusion of m_2 into m_1 using the approximate translation and tuned rotation: $trf = [x \ y \ \theta]^T = [43 \ -13 \ 19.8^\circ]^T$.

After applying the proposed tuning method for translation, the tuned transformation becomes $trf = [x \ y \ \theta]^T = [40 \ -16 \ 19.8^\circ]^T$. Fig. 4.17-b shows the exact alignment for m_2 into m_1 . The verification index after final alignment is 97%.

The next step is to fuse m_1 with m_3 . Similarly Fig. 4.18-a shows results of map

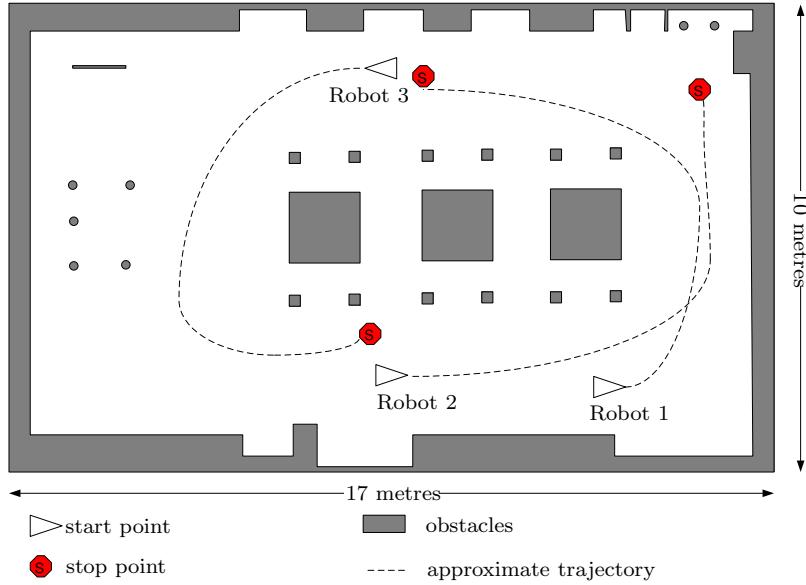


Figure 4.15: Floor plan of the real world test environment. Start and stop points of the robots are marked with triangles and stop signs, respectively. The trajectory of each robot is depicted with dashed lines and obstacles are shown in dark colour. All markings are approximate.

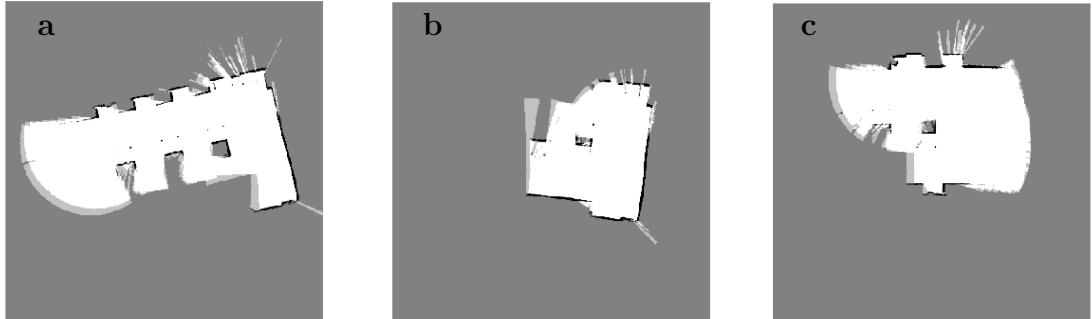


Figure 4.16: Three local maps provided by three robots based on the approximate dimensions and trajectories of Fig. 4.15. **a)** m_1 provided by $robot_1$, **b)** m_2 provided by $robot_2$, **c)** m_3 provided by $robot_3$

fusion with the approximate translation and tuned rotation. Fig. 4.18-b shows the result of the tuned alignment. Approximate and exact translations for the fusion of m_1 and m_3 are $[-74 \ 23 \ 192.8^\circ]^T$ and $[-67 \ 32 \ 192.8^\circ]^T$, respectively. After the final alignment the verification index is 98%.

From the Fig. 4.17 and Fig. 4.18, it is clear the “approximate” maps are quite accurate. This is generally the case. However, if the approximate rotation or translations are

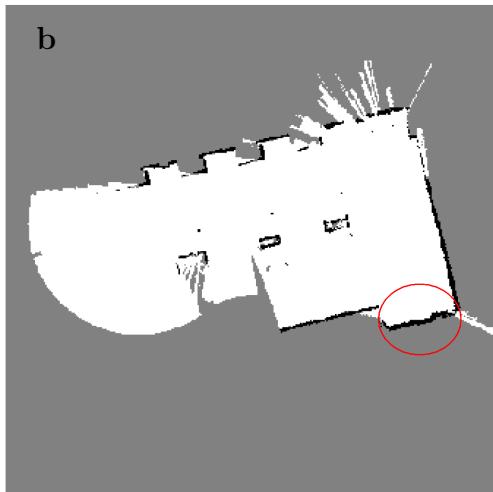
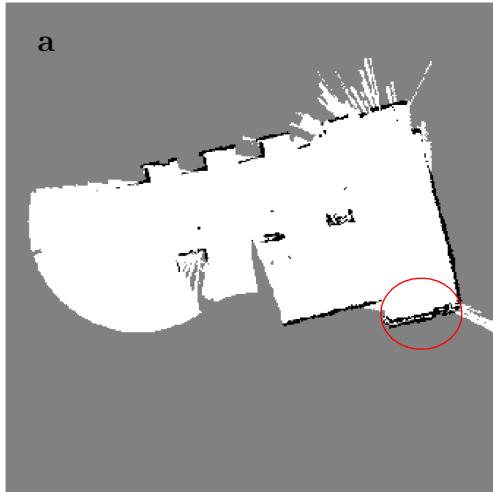


Figure 4.17: Map fusion for m_1 and m_2 based on the local maps presented in Fig. 4.16-a, b **a)** alignment of two maps after approximated translation and tuned rotation, **b)** alignment of two maps after exact transformation. Differences between the approximate and tuned maps are highlighted.

highly inaccurate, it can result in higher computation time in the tuning step due to the increased search space. Finally, Fig. 4.19 shows all three local maps fused together and shown in the local coordinates of m_1 .

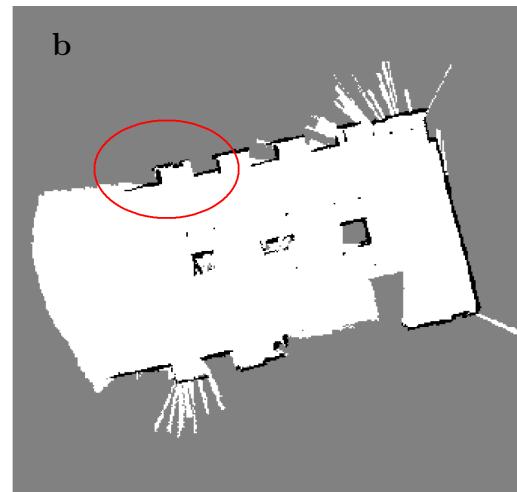
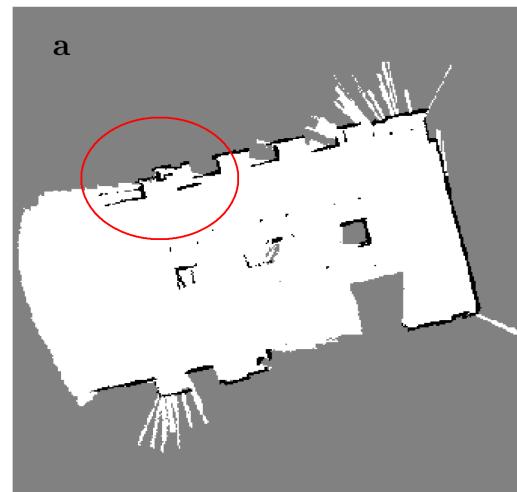


Figure 4.18: Map fusion for m_1 and m_3 based on the local maps presented in Fig. 4.16-a, c **a)** alignment of two maps after approximated translation and tuned rotation, **b)** alignment of two maps after exact transformation. Differences between the approximate and tuned maps are highlighted.

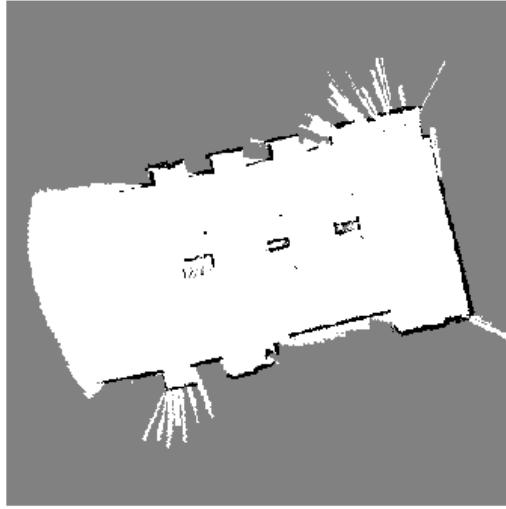


Figure 4.19: Map fusion for m_1 , m_2 and m_3 based on the local maps presented in Fig. 4.16-a, b, c; m_2 and m_3 are transformed to the coordinates of m_1 after determining the tuned transformation matrices

It is clear from the results that the three robot team is able to map the environment more quickly than one robot alone. Each robot in the team generates a map of some part of the workspace and then, if there is some overlap between the maps, all local maps can be fused into a global map. It is important to note that the robots are not required to meet at any point to determine their relative positions, this information is generated from the maps themselves.

4.2.6.3 Discussion: Tuning Parameters

Throughout the algorithm, many parameters are introduced which need to be tuned for good performance. Table 4.1 lists these parameters. In this work, these parameters have been tuned by trial and error; however, it is possible to use non-derivative optimization methods, such as genetic algorithms or particle swarm optimization, which is beyond the scope of the work. Repeated experiments show that changing these nominal values for about $\pm 10\%$ does not affect the quality of the output significantly. The values of the t_h , mentioned in the first row of Table 4.1, is relative to

the maximum of the gradient magnitude of the map, defined in equation (4.4) and calculated online during the edge detection process.

Table 4.1: List of important parameters with their nominal values.

Block	Parameter	Value
Edge Detection	t_h : high threshold	$\max(E_s)$
	t_l : low threshold	$0.4 t_h$
Edge Smoothing	d_t : smoothing threshold	10
Segmentation	n : number of points in each segment	50
Segment Verification	h_d : distance histogram threshold, (4.15)	3
	ω_c : discontinuity threshold, (4.16)	6
	number of bins of a histogram	7
Pose Verification	verification threshold	94 %

4.2.6.4 Discussion: Comparison Results

The proposed method has been compared with adaptive random walk (ARW) map merging [2]. ARW is based on search and verification. In ARW, the similarity indices of two maps are calculated given a set of transformations. The set is composed of known rotations and translations. The transformation corresponding to the best similarity index is selected as the start point of an adaptive random walk search to merge the maps more accurately. Finding the initial transformation from the set takes the majority of the processing time. To compare the results with ARW, a faster version of ARW is used where a set of 144 transformations have been investigated. In this case the proposed algorithm runs at least ten times faster.

Table 4.2 summarizes the comparison of the processing times for the two experiments. To perform an accurate comparison, all computations were performed on a Core2Duo 2.66 GHz laptop. In the first experiment, the algorithm takes about 84 seconds to run. This is about half of the time that the algorithm presented in [2] required to run on the identical environment producing similar results. For the second experiment, the entire algorithm required about 95 seconds to run, which is less than the time

required for ARW.

Table 4.2: Comparison of processing times

Method	Experiment 1	Experiment 2
Map segmentation	≈ 84 s	≈ 95 s
ARW map merging [2]	≈ 152 s	≈ 160 s

Table 4.3 compares verification indices of the these two methods for the two experiments. In the first experiment the indices are close but in the second experiment, the proposed method provides a better verification index.

Table 4.3: Comparison of verification indices

Method	Experiment 1	Experiment 2
Map segmentation	$\approx 95\%$	$\approx 98\%$
ARW map merging [2]	$\approx 95\%$	$\approx 94\%$

In experiment 1 the amount of overlap was approximately 35%, in experiment 2 the average overlap of the original maps was approximately 25%. However, it is important to note that success or failure of the map merging process is more dependent on the existence of matchable segments within the overlapping sections of the map than on the actual size of overlapping sections of the maps.

The feature extraction method proposed here based on segment matching is particularly well-suited to matching occupancy grid map structures. Other feature extraction methods designed to work on imagery, such as Fourier transform, were tested and produced poorer results.

It should be stated that, in this work, the paths of the robots were planned *a priori* with the real focus of the approach being on map merging. In future studies we will explore how to combine the proposed map merging approach with path planning for efficient real time exploration and localization.

In addition, in this work maps are aligned only once to find the relative transformation between the two robot poses. Once this transformation is known, it is assumed that

further explored parts of the map can be easily fused. It would be interesting to explore how the performance could be improved if the map merging process happens repeatedly as the robots explore the environment.

4.2.7 Summary

In this work, a new method of multiple-robot SLAM is developed. The proposed method is based on improvements made to single-robot SLAM algorithms. Results are verified with tests performed in real environments. Novel aspects of this approach include, preprocessing of occupancy grid maps using Canny Edge detection and smoothing, segmentation using a novel histogram method to find unique geometrical characteristics, cross correlation to find matching patterns in maps, methods for determining approximate and exact transformation matrices that relate the maps, and verification. A major advantage of this approach is that there is no need for the robots to meet one another to determine their relative positions. They can do localization and mapping independently without relying on tracking one another.

When the relative transformations between the robots are known, maps and poses should be updated. Moreover, the uncertainty of the relative transformations should also be taken into account. In Section 4.4, updating maps and the effect of the uncertainty of the relative transformations on the transformed maps will be described. The effect of the uncertain transformations on the poses will be studied in Chapter 5. In the future, improving the pose extraction by using better search algorithms will be investigated. Incorporating tracking information can also improve the results, meaning that if the robots do happen to see each other, they can find their relative poses quickly and easily and use them to generate more accurate results.

4.3 Neural Networks Approach

In this section, another solution for SLAM with multiple robots is developed. Each robot performs single robot view-based SLAM using an extended Kalman filter to fuse data from two encoders and a laser ranger. To extend this approach to multiple-robot SLAM, a novel occupancy grid map fusion algorithm is proposed.

The proposed map learning method is a process based on the self organizing maps (SOM). In the learning phase, the obstacles of the map are learned by clustering the occupied cells of the map into clusters. The learning is an unsupervised process which can be done on-the-fly without any need to have output training patterns. The clusters represent the spatial form of the map and make further analyses of the map easier and faster. Also, clusters can be interpreted as features extracted from the occupancy grid map so the map fusion problem becomes a task of matching features. Results of the experiments from tests performed on a real environment with multiple robots prove the effectiveness of the proposed solution.

4.3.1 Problems with Existing Methods and Motivations

All of the previously published map merging methods are so computationally intensive that the robots will have to move very slowly for real-time operations. A more computationally efficient approach will result in the robots being able to move more quickly through the environment and, consequently, achieve the task in shorter time. Neural network algorithms are powerful artificial intelligence solutions which can be applied to any problem involving learning or training. In multiple-robot SLAM there is the need to learn a map so neural networks are a suitable approach. After learning a map, any integration with other maps will be much faster and simpler due to the reduced dimensionality.

Many different neural network architectures exist. Feed-forward networks are popular

due to their simple structure; however, the training methods require supervision, or input and output patterns, to train the network. Since the map is unknown, these input-output pairs are not available. Kohonen networks, or self organizing maps (SOM), which have a recurrent structure, use unsupervised training to reduce the dimensionality of input data. This means that only input patterns are required to train the network. This type of network is a good candidate to solve the map fusion problem because the weights and parameters of the network are calculated without any need to specify output patterns. The popularity of SOM in engineering applications is due to its ability to train without supervision, small developmental effort required, and adequate rejection of outliers [117].

There is no known literature that applies neural networks to the multiple-robot SLAM problem. The proposed method is motivated by the fact that developing a fast and reliable data processing method for a team of multiple robots is very important for both accuracy and scalability of the result. A multi-step algorithm is proposed where the SOM is used to extract important information from the map for faster data processing in the subsequent steps.

4.3.2 Background: Self-Organizing Maps

In this section, self-organizing maps are briefly introduced and the learning procedure is defined mathematically.

The SOM is a powerful neural network paradigm which can detect internal correlations of its input vectors and categorize them according to their similarity. The SOM is an orderly mapping of a data set with high-dimensional distribution onto a low-dimensional structure. It is mainly used for the visualization and abstraction of large data sets [118].

The mapping of SOM preserves the topologic-metric relations between input data. The mapping projection is performed using a matching process where a generalized

model is assumed to be associated with each grid location. For each input item, the closest model is identified. Finally the collection of models is optimized such that all inputs are approximated [119].

The SOM is constructed based on the competitor networks where inhibitory and excitatory effects of neurons on each other provide the learning infrastructure. An SOM is trained using the Kohonen learning rule and the inputs are presented in random order [120]. The training process starts with identifying the winning neuron for each given input vector. Then weight vectors in the neighbourhood of the winner neuron are set to the average position of all input vectors. The process is performed for a specified number of iterations.

Consider an input sample, $x(k) \in \mathbb{R}^2$, where k is the sample index, with the last weights computed for the i^{th} neuron as $w_i(k) \in \mathbb{R}^2$. When a new input sample is applied, the improved weights of the neuron are determined using a recursive rule

$$w_i(k+1) = w_i(k) + h_i(k)(x(k) - w_i(k)), \quad (4.29)$$

where h_i is the neighbourhood function. The neuron with the minimum distance is called the winner [118]. More detailed training methods are found in [120].

4.3.3 Description of the Proposed Method

Two important issues with multiple-robot SLAM are finding the relative initial poses of the robots and coping with the uncertainty of the maps of the robots. Since the maps are considered to be occupancy grid maps, the fusion algorithm should take into account the associated uncertainties. In other words, more emphasis should be given to the cells with lower uncertainties of occupancy. In this research, the map learning process is performed by clustering the map into specific clusters using unsupervised Kohonen networks. The clusters represent the main characteristics and features of

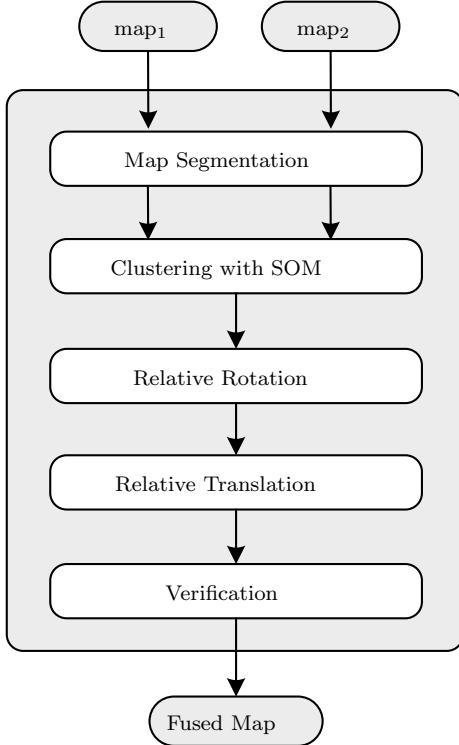


Figure 4.20: Flow chart of the proposed algorithm.

the map and make further map processing easier and faster.

The main goal of the proposed method is to find the relative transformation between maps as quickly and reliably as possible, while considering the uncertainties. An overview of the method is shown in Fig. 4.20. The inputs of the algorithm are the two occupancy maps and the output is their relative transformation.

4.3.3.1 Map Segmentation

The first operational layer that is performed is map segmentation. This preprocessing step helps to produce better results in the remaining steps. The motivation for this preprocessing is the observation that the map is usually composed of a few major segments which are relatively far from each other. The segmentation identifies discontinuous segments of obstacles located far enough from each other that they should be considered as separate. This is necessary so that the SOM algorithm can

be run on each contiguous segment of obstacle in the map. Otherwise, the SOM will produce clusters that are not close to obstacles but could be the midpoint between two separate obstacles.

The pseudo-code for map segmentation is given in Algorithm 4.3.1. In line 2, M is defined as the set of all unclustered obstacle points in the occupancy grid map. A random point, $p_0 = [x_0, y_0]^T$, called a *start point* is selected from M in line 5. The set, S , is defined as the set of all start points. The set that contains all points in a segment i is $C_i, i = 1..n$. The algorithm continues for each segment as long as there are start points remaining in S to be processed.

For each start point, $p_s = [x_s, y_s]^T$, the set of neighbouring points, C_s , is determined in line 9. The set of potential new start points, S_s is calculated in lines 10 and 12, where the ring radius begins as 1, and then increases to Δr if no results are found. The reason for doing this process in two steps is to try to limit the number of potential next start points as much as possible to increase the speed of the algorithm.

In line 14, the set of next start points is updated to include the members of S_s that are not already in the segment C_i , and the current start point, p_s is removed. The segment, C_i , is updated to include the values of C_s and S_s .

Once S is empty, the segmentation is finished. The points in the segment are removed from M and i is increased. The process continues until M is empty. Fig. 4.21 shows a graphical representation of the process.

The extracted segments of each map are passed to the next level where the SOM is applied on each segment to cluster it. Segments with small size do not provide enough information for map fusion, so they are not used for clustering.

4.3.3.2 Clustering with Self-Organizing Map

Once the map segmentation is complete, the clustering is performed on each segment. The clusters, which are generated by applying SOM, will be used to fuse the two maps

Algorithm 4.3.1 Map Segmentation

Input: Set of occupied cells from occupancy grid map: M_{occ}

Search radius: r

Search radius increase step: Δr

Output: n set of segments: $C_i, i = 1, \dots, n$

```
1:  $i \leftarrow 0$ 
2:  $M \leftarrow M_{occ}$ 
3: while  $M \neq \emptyset$  do
4:    $i \leftarrow i + 1$ 
5:    $p_0 \leftarrow$  random element of  $M$ .
6:    $S \leftarrow \{p_0\}$ .
7:   while  $S \neq \emptyset$  do
8:      $p_s \leftarrow [x_s, y_s]^T \leftarrow$  random element of  $S$ 
9:      $C_s \leftarrow \{p = [x, y]^T | (x - x_s)^2 + (y - y_s)^2 \leq r^2\}$ 
10:     $S_s \leftarrow \{p = [x, y]^T | r^2 < (x - x_s)^2 + (y - y_s)^2 < (r + 1)^2\}$ 
11:    if  $S_s = \emptyset$  then
12:       $S_s \leftarrow \{p = [x, y]^T | r^2 < (x - x_s)^2 + (y - y_s)^2 < (r + \Delta r)^2\}$ 
13:    end if
14:     $S \leftarrow S \cup (S_s - C_s) - p_s$ 
15:     $C_i \leftarrow C_i \cup C_s \cup S_s$ 
16:   end while
17:    $M \leftarrow M - C_i$ 
18: end while
```

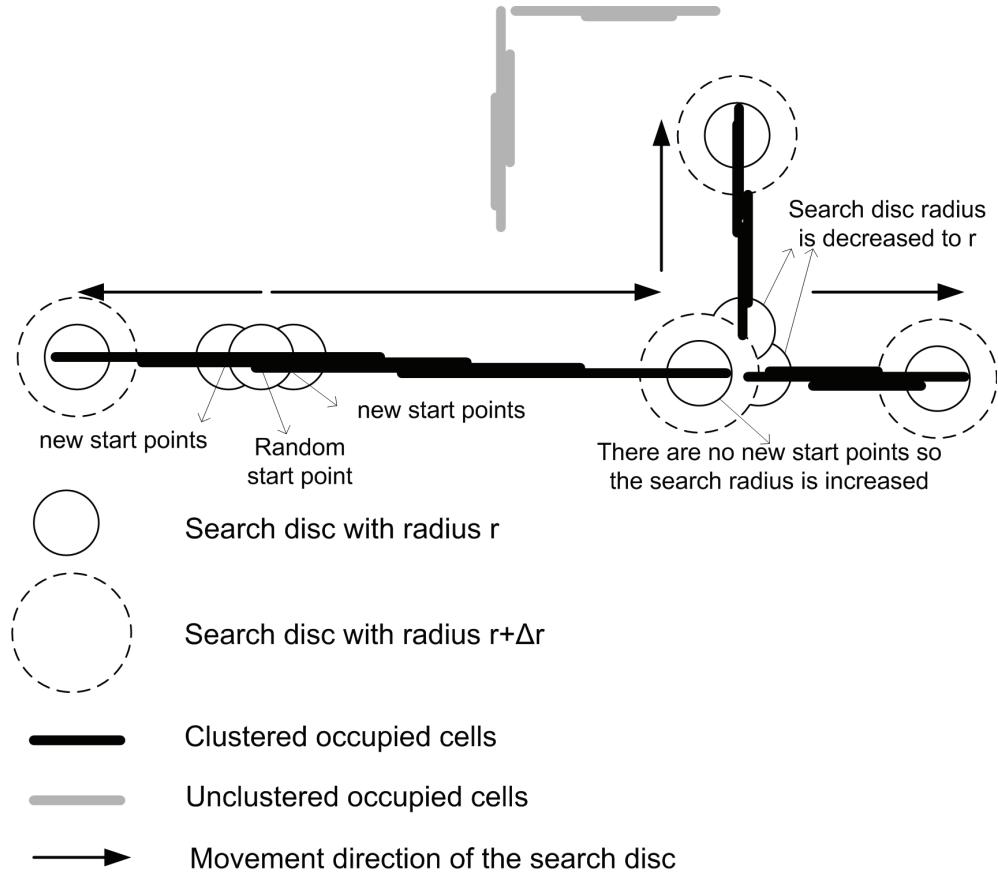


Figure 4.21: Map Segmentation as described in Algorithm 4.3.1.

(refer to Fig. 4.20). An occupancy grid map is composed of a large number of cells which are time consuming to process individually. The purpose of performing this clustering is to decrease the time required for map fusing. The clusters preserve the original information or shape of the obstacles or features from the map but represents them in a much more compact form.

Fig. 4.22 shows the simple structure of the network. In this case the data is a two dimensional position of an occupied cell in the occupancy grid map. This should not be confused with the locations of the neurons in the solution space, which are also two dimensional locations. In the training phase, the location of each occupied cell is presented as the input of the network. The network finds the best matching or winning neuron to be the one whose weights, or data, are closest to that input

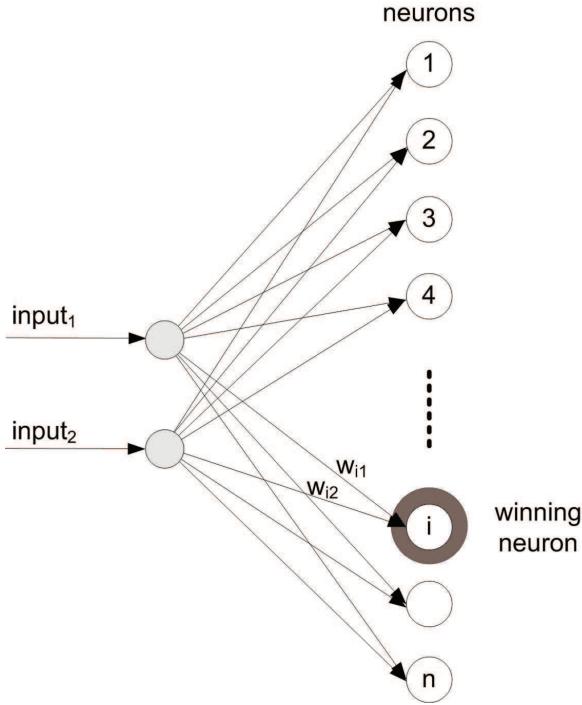


Figure 4.22: Structure of the SOM.

position. The weights corresponding to that neuron and the neurons that are in the neighbourhood in the solution space are updated to better represent that input pattern. Once the algorithm is run for several iterations, the final weights are the clusters that represent the data.

The number of neurons or clusters, n is an important design parameter. Its selection is dependant on the size of the map, and it is noted that the choice of n has a significant effect on the quality of the results. In this work the number of neurons is assigned proportionally to N , the number of the occupied cells in the map.

$$n = \text{round} \left(\frac{N}{\eta} \right), \quad (4.30)$$

where η is a scaling gain.

It is emphasized that the training process and then subsequent map fusion are fast enough that they can be implemented onboard a robotic platform.

A major advantage of the occupancy grid map representation is the treatment of uncertainty. Values in an occupancy grid map are float values in the range $(-1, 1)$ where negative values indicate belief of an obstacle, and positive values indicate belief that the cell is free. At first, all cells are initialized to 0, indicating that the state of the cell is unknown. As sensor data is processed, the values in the map are adjusted. For example, if a cell is detected as being occupied by many scans, it will have a value closer to -1 than if it had only been detected as occupied a small number of times. In order to account for this uncertainty, input patterns with higher certainty of being obstacles are presented more frequently as the input in the training of the SOM. In a very simple occupancy grid map, each time a cell, c , is detected as occupied, the value is decreased by some tunable parameter Δ to a minimum of -1 . Alternately, each time c is detected as free, its value is increased by Δ , to a maximum of 1 . Only cells that have negative values (obstacles) are used in the training of the SOM. The relative frequency that each input sample is presented is determined by the relation:

$$f(c) = \text{round} \left(\frac{|c|}{\Delta} \right), \quad (4.31)$$

where the ratio of $|c|$ to Δ represents the number of times that the cell was detected as an obstacle. This value is rounded such that the cell c appears in the training phase with a frequency of $f(c)$. For example, if $\Delta = 0.1$, a cell with a value of -0.4 should appear as the input to the SOM four times more often than a cell with a value of -0.1 .

Once the clustering has been performed, the relative transformation between the maps can be found efficiently. This involves extracting the relative orientation and translation separately as will be discussed in the next sections.

4.3.3.3 Relative Orientation

The arrangement of the cluster points by the SOM represents the approximate map of the world. The surface which is created by connecting the clustered points is a representation of the real surface of the world. Norm vectors are a way of mathematically representing a surface that is defined by points. By comparing the norm vectors of two maps, it is possible to determine if they are a match, and then subsequently extract their relative orientation. Fig. 4.23 shows an illustration of four cluster points. The surface generated by connecting adjacent clusters is termed the cluster surface (dotted line). The cluster surface norms are unit vectors that are perpendicular to the cluster surface and are equidistant from two clusters. The relative orientation between the two maps is determined by performing a 360° histogram on the directions of the cluster surface norms, and then matching the histograms of the two maps. Some sample histograms are shown in the experimental results section. This is similar to the circular cross correlation of histogram norms that is presented in [15] for aligning two consecutive local maps. However, because the number of data points has been effectively reduced through clustering, the size of the buckets in the histogram must be relatively large. As a result, it is not possible to get sufficiently accurate results. A tuning method based on the Radon transform is used to find a more accurate value for the relative orientation [121].

The proposed solution for tuning using the Radon transform, which was introduced in Section 4.2, is also used here. The tuned angle is represented as

$$\alpha = \alpha_{app} + \delta_{tuning}, \quad (4.32)$$

where α is the tuned rotation angle, α_{app} is the result of the cross correlation, δ_{tuning}

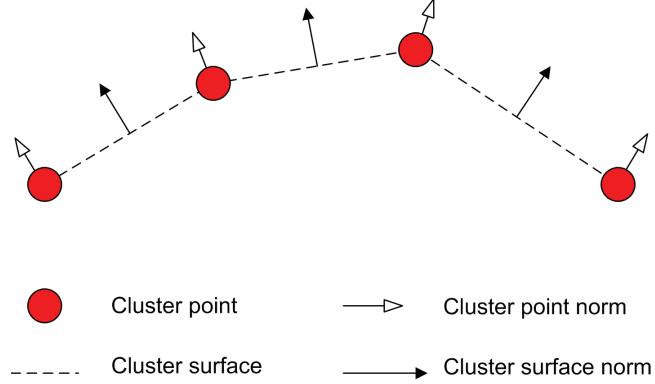


Figure 4.23: Surface norm is the perpendicular vector to the surface and the point norm is the average norm of the two line segments connecting the point to two adjacent points. If the point is connected to just one point, the point norm is the same as the surface norm.

is the tuning angle added to α_{app} to adjust it and is defined as

$$\delta_{tuning} = \underset{\theta}{\operatorname{argmin}} (\mathcal{R}_{\theta=0:180}(m_1)) - \underset{\theta}{\operatorname{argmin}} (\mathcal{R}_{\theta=0:180}(T_{\alpha_{app}}(m_2))), \quad (4.33)$$

where $\mathcal{R}_{\theta=0:180}(map)$ is the Radon image of the input map over the specified domain of angles (θ). The operation $T_{\alpha_{app}}(map)$ is a rotation of the input, map , by α_{app} and $m_1 = map_1$ and $m_2 = map_2$. The first $\operatorname{argmin}(\cdot)$ yields an angle corresponding to a straight line of cluster-points in m_1 . $T_{\alpha_{app}}(m_2)$ rotates m_2 according to the extracted approximate rotation angle. Now the second $\operatorname{argmin}(\cdot)$ gives the angle corresponding to straight line of cluster-points of the rotated m_2 . The difference between these two angles is the tuning angle to be used in equation (4.32).

Once the tuned relative orientation has been found, it is applied so that the relative translation can be found as will be discussed presently.

4.3.3.4 Relative Translation

Once the relative rotation has been determined, it is highly probable that points with a similar orientation as determined by the norm vectors actually correspond to the same point in the world. This fact is used to do an efficient search to associate points

from different maps to each other by finding their relative translation. Referring again to Fig. 4.23, the cluster point norms are unit vectors that originate from the clusters at an angle that is the average of angles of the two adjacent cluster surface norms. For cluster points that are at the end of the cluster surface, the point norms are parallel to the surface norms. These point norms are used to find the relative translation between the two maps using an iterative approach similar to the iterative closest point (ICP) Algorithm [122]. The pseudocode for the algorithm is presented in Algorithm 4.3.2.

In Algorithm 4.3.2, the inputs are $M_1^{1 \times k_1} = \{m_i^1\}_{i=1}^{k_1}$, and $M_2^{1 \times k_2} = \{m_i^2\}_{i=1}^{k_2}$, the sets of cluster points for *map*₁ and *map*₂ respectively after the relative rotation has been applied, and the associated sets of point norms $N_1^{1 \times k_1} = \{n_i^1\}_{i=1}^{k_1}$, and $N_2^{1 \times k_2} = \{n_i^2\}_{i=1}^{k_2}$ (the norm of m_i^1 is n_i^1 for all i . Similar for M_2). Without loss of generality, M_2 is being translated to M_1 .

In line 5, for every point in M_2 , a new set, P_k ($P_k \subseteq M_1$, $k = 1, \dots, k_2$) is built that contains the points in M_1 that have norms within a matching angle threshold, ϵ of n_k^2 (the point norm of m_k^2). If this set P_k is non-empty, then the point in P_k that has the smallest Euclidean distance from m_k^2 is chosen as the winner and a correspondence is made. These correspondences are maintained in two new sets, PT_1 and PT_2 , as shown in lines 7 and 8. If the set P_k is empty, it means that there were no points in M_1 with similar norms, or that this part of *map*₂ has no matching part in *map*₁, so no correspondence is made and the algorithm continues.

Once all of the points in M_2 have been considered, the best translation is found (lines 12-14). Conceptually, δ_x and δ_y are the means of errors between corresponding matching points from the two maps along the X and Y axes respectively.

Once the translation is found, it is applied to the set M_1 , the matching angle threshold is reduced and the algorithm is restarted. The stopping criterion is either a number of iterations or a minimum error threshold is reached between the two cluster sets

Algorithm 4.3.2 Translation realization based on the norms of the cluster points

Input: Sets of clusters M_1 and M_2 with associated sets of point norms N_1 and N_2
The matching angle threshold: ϵ
The maximum number of iterations: $iterations_{max}$
The error threshold: J_{thresh}

Output: Translation, tr

```
1:  $iterations \leftarrow 0$ 
2: repeat
3:    $i \leftarrow 1;$ 
4:   for  $k = 1 \rightarrow k_2$  do
5:      $P_k \leftarrow \{ \text{All points } p \mid [x, y]^T |p| = m^1 \in M_1 \text{ whose associated norm } n^1 \text{ satisfies } |n^1 - n_k^2| < \epsilon \}$ 
6:     if  $P_k \neq \emptyset$  then
7:        $PT_1[i] \leftarrow \text{element of } P_k \text{ that is closest to } m_k^2$ 
8:        $PT_2[i] \leftarrow m_k^2$ 
9:        $i \leftarrow i + 1$ 
10:      end if
11:    end for
12:     $\delta_x \leftarrow \frac{1}{i-1} (\sum_{l=1}^{i-1} PT_{1x}[l] - \sum_{l=1}^{i-1} PT_{2x}[l])$ 
13:     $\delta_y \leftarrow \frac{1}{i-1} (\sum_{l=1}^{i-1} PT_{1y}[l] - \sum_{l=1}^{i-1} PT_{2y}[l])$ 
14:     $tr \leftarrow \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix}$ 
15:    Each point in  $M_2$  gets shifted by  $tr$ 
16:     $J = \sum_{l=1}^{i-1} ||PT_1^l - PT_2^l||$ 
17: until  $J < J_{thresh}$  or  $iterations > iterations_{max}$ 
```

M_1 and M_2 .

It is noted that the selection of the matching angle threshold, ϵ , is quite important. Values that are too large give inaccurate results and values that are too small do not yield any correspondences. An effective tradeoff is to reduce the value of ϵ at every iteration. In this case it was initialized to 25° and reduced gradually until it reached a minimum of 4° .

4.3.3.5 Verification

A final verification process is used to eliminate false matching or to select the best result of a few candidates. The verification method is based on the convergence of the performance index, J as given by:

$$J = \sum_{i=1}^n ||p_i^1 - p_i^2||, \quad (4.34)$$

where n is the number of clusters, and p_i^1 and p_i^2 are the corresponding cluster points for m_1 and m_2 respectively. J is the sum of the squared Euclidean distances between corresponding matching points. To have a perfect matching, J should be a small number. J is calculated for each iteration of the relative translation calculation. If J converges then the fusion of the clusters can be considered as valid and the next verification method ensures the accuracy of the results.

In order to ensure that the entire maps have been accurately fused, the verification index, introduced in Section 4.2, is used. The verification index gives a measure of how similar the two maps are. A similarity index of 100% would mean that the two maps are identical.

4.3.4 Advantages and Disadvantages

The proposed solution learns the maps by artificial intelligence techniques. The map learning clusters the map into many features. This process reduces the scale of the map and hence makes further processing faster. The experimental results will compare processing time of the proposed algorithm with two other algorithms and show its efficiency. Reducing the scale of the map is at the cost of losing information in the original scans. Moreover, the clusters are an approximate representation of the map, and compared with the map, clusters are less accurate.

4.3.5 Discussion

The capability for supporting multiple agents is one concern of any robotics algorithm. As mentioned in Section 4.3.3.2, an important advantage of the SOM clustering method is a down-scaling of the grid map into cluster points. This down-scaling

reduces the required time for map fusion and enhances scalability. The training phase of the SOM is computationally intensive, but time savings in the map fusion make it worthwhile. This allows more robots to be added to the team while still operating efficiently. The down-scaled map includes the most important information and any loss of information will be accounted for by fine tuning with the Radon transform. Deploying a parallel hierachial structure among the members of a team can save tremendous amount of time. For example, in a group of six robots, robots 1 and 2 can fuse maps and submit the result to robot 3. Simultaneously, robots 4 and 5 can do the same and submit the result to robot 6. Then the global map is obtained from the map fusion from robots 5 and 6. The result should be submitted to all robots in the team. The formation of the robots to generate this hierachial structure is an interesting problem which is left for future work.

4.3.6 Contribution

In this work a new method of map fusion for multiple-robot SLAM is introduced. A new algorithm is proposed which is based on the neural network theory. In this new approach, the map is learned by clustering, which reduces complexity and increases speed and accuracy. The maps are fused into a global map using relative transformation matrices determined using the clustered points.

To summarize, the contribution of this research is a layered processing algorithm including:

- A high level map segmentation algorithm for occupancy grid map preprocessing.
This step prepares the map for SOM clustering.
- Application of SOM to cluster or learn the preprocessed map. By clustering, important features of the map are extracted into cluster points, which are used for map fusion. The motivation for clustering using SOM is its unsupervised

training ability, and the accuracy and speed of the response.

- Inclusion of map uncertainty in the learning phase. The map is built using sensors that have inherent noise. This noise generates uncertainty in the resulting map and should be considered during the clustering process.
- Estimation of the relative transformation matrix of two maps using the cluster points. The cluster points are used to extract the relative orientation and translation of the maps.
- The use of surface norms to determine which cluster points from two different maps should be selected to determine the relative transformation.

4.3.7 Experiment

Two experiments are performed in real-world indoor environments with two robots. The first experiment is performed in a simple environment and the second one is done in a more complex environment. The tests were performed by CorBot robots.

4.3.7.1 Case1: Simple Test Environment

Fig. 4.5 shows the approximate floor plan and dimensions of an indoor environment used for the experiment. This room has been surveyed by two robots from different initial positions which are assumed to be unknown.

Fig. 4.24-a shows the local map provided by the first robot, and Fig. 4.25-a shows the local map provided by the second robot.

After applying map segmentation to both maps, Fig. 4.24-b and Fig. 4.25-b show the different map segments in different colours. Since small segments do not carry enough information for map fusion, only the larger segments from each of the maps are matched. Runs of the map segmentation on a 2.99 GHz dual core Intel based computer yielded an average run-time of about 1 second. The results of the clustering

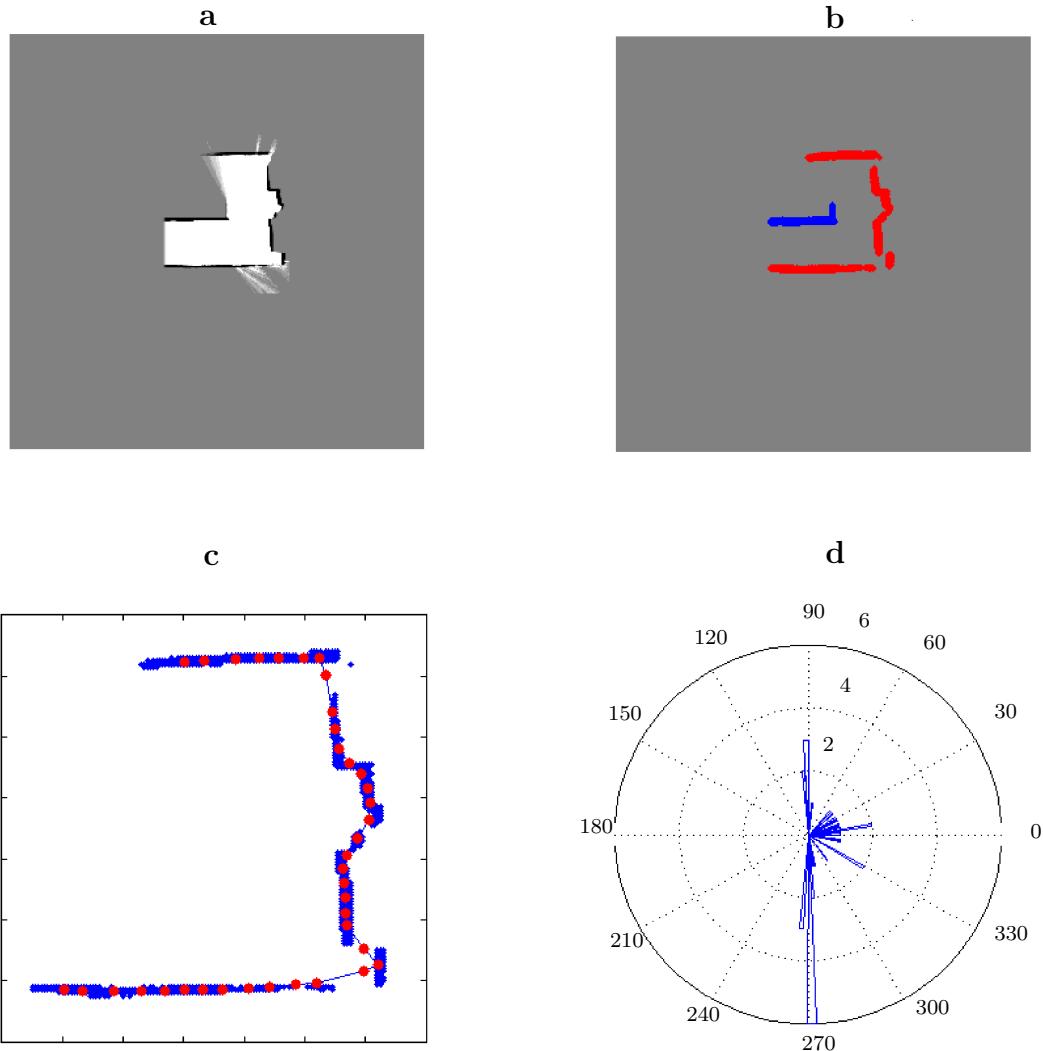


Figure 4.24: **a)** m_1 , occupancy grid map of the test environment provided by the first robot. **b)** Map segmentation in different colours developed using Algorithm 4.3.1. **c)** Clusters developed using SOM. **d)** Histogram of the norm vectors.

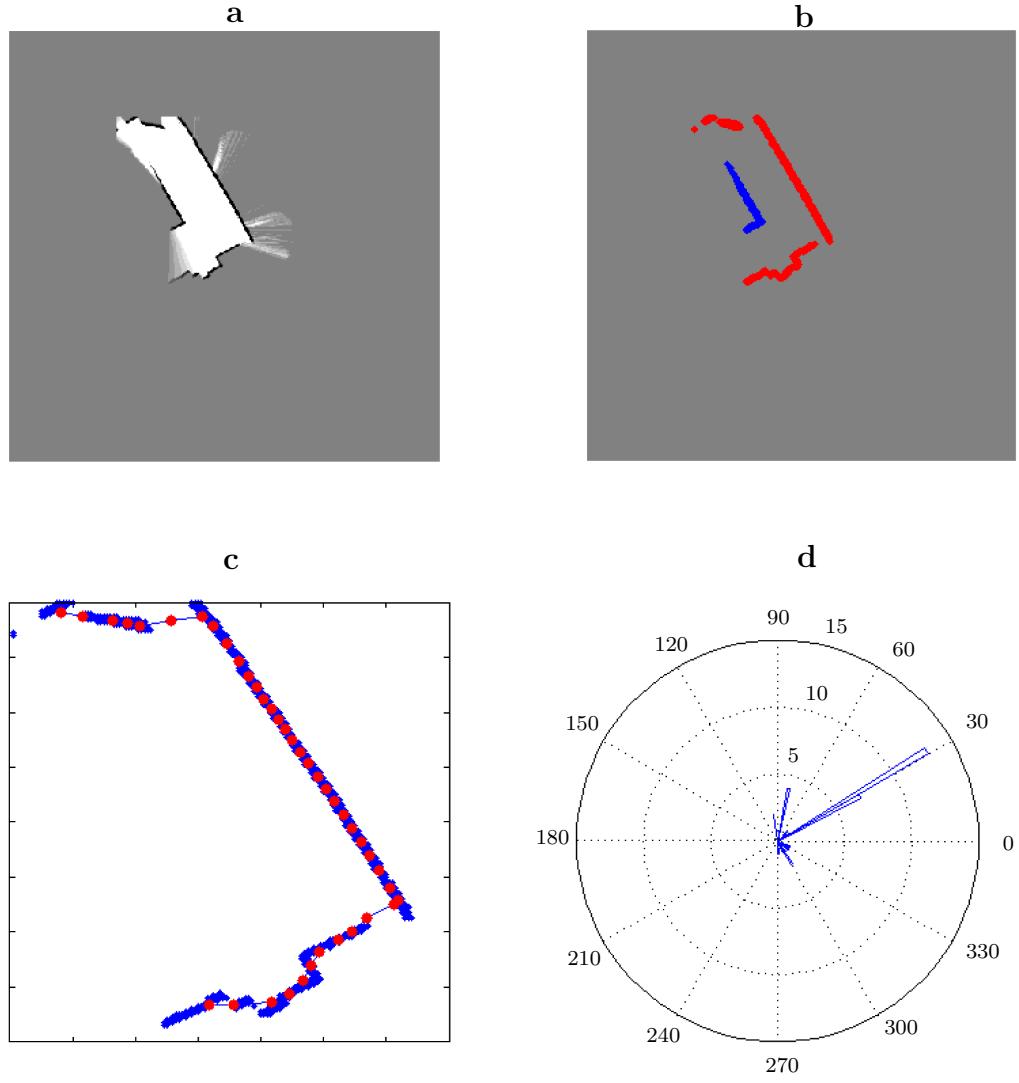


Figure 4.25: **a)** m_2 , occupancy grid map of the test environment provided by the second robot. **b)** Map segmentation in different colours developed using Algorithm 4.3.1. **c)** Clusters developed using SOM. **d)** Histogram of the norm vectors.

using SOM are shown in Fig. 4.24-c and Fig. 4.25-c. Clusters tend to keep the spatial characteristics of the map, so any transformation which can fuse the clusters is a good estimate for the fusion of the maps. The training of the SOM is the most time consuming step of the process. Referring to equation (4.30), m_1 has 685 occupied cells and m_2 has 715. Assuming η to be 18, 38 neurons are assigned for m_1 and 40 for m_2 . η is chosen by experiment. For 40 cluster points, the average time for training is about 8 seconds and the result converges after 5 iterations. Fig. 4.24-d and Fig. 4.25-d depict the normals histogram of the clustered points. Applying circular correlation to the orientation histograms, gives the relative orientation of 57.5 degrees. Fig. 4.26-a shows the clusters after the rotation by the cross correlation. Fig. 4.27-a and Fig. 4.28-a show the Radon images for m_1 and m_2 respectively after applying the approximate transformation. As Fig. 4.27-b and Fig. 4.28-b show, the peak point of the Radon image for m_1 happens to be at 91 degrees while for the rotated m_2 it is 93 degrees. The difference of 2 degrees is for tuning. Fig. 4.26-b shows the clusters after rotation tuning by the Radon transform. Using the proposed iterative approach, translation is found. In Fig. 4.26-c, the translated clusters are shown. The translation vector is calculated as [49.3, 59.8] along X and Y axes. Finally, Fig. 4.26-d shows both maps fused using the extracted rotation and translations. Fig. 4.29 shows the performance index defined in (4.34). The error converges after 15 iterations. The algorithm may be stopped at this point. The verification index, $V(m_1, m_2)$ defined in Section 4.27 is 95%, which shows that the two maps are indeed quite well matched.

4.3.7.2 Case2: More Complex Environment

To demonstrate the effectiveness of the proposed methods, another experiment is performed in a larger environment with the approximate size of 17×10 metres. Fig. 4.30 shows the approximate floor plan of the test environment with the paths of the robots. This experiment is similar to the experiment presented in Section 4.2.6.2,

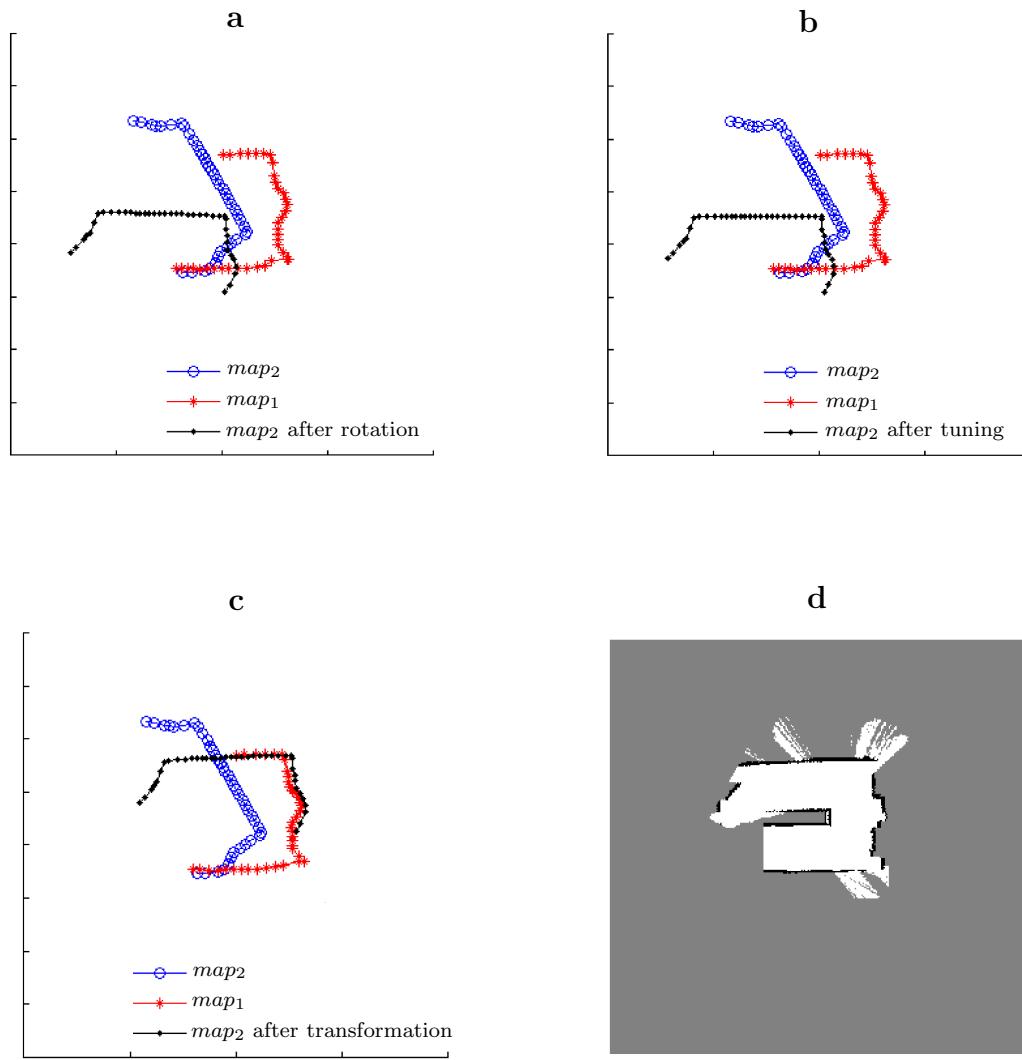


Figure 4.26: **a)** Both maps and m_2 after the rotation. **b)** Both maps and m_2 after applying the rotation and the translation. **c)** m_1 and m_2 after applying the extracted relative transformation.

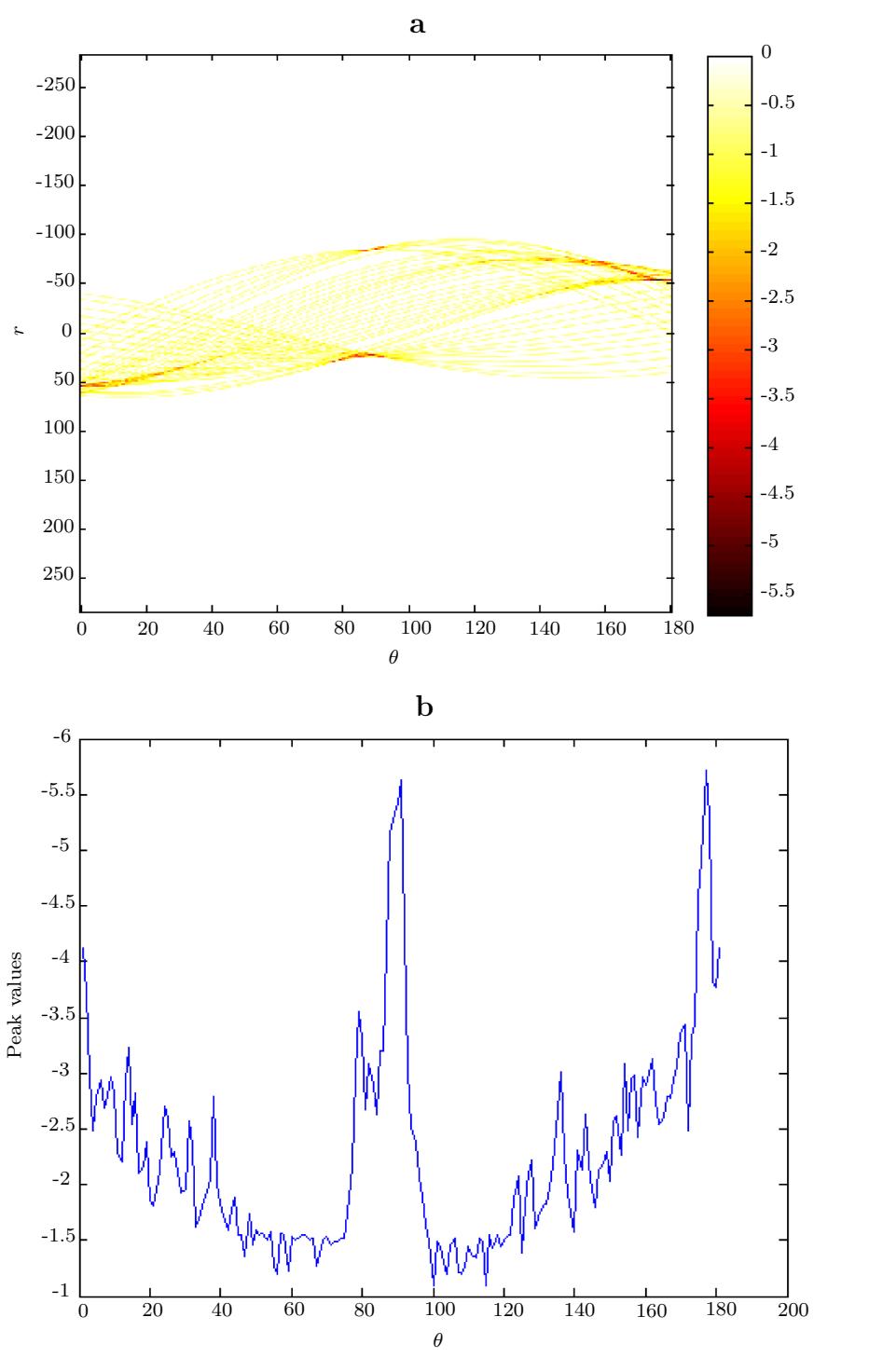


Figure 4.27: **a)** Radon image of m_1 over 180 degrees **b)** peak point of the Radon image

but in this experiment only two robots have been used. As mentioned in Section 4.3.4, the proposed algorithm in this section is faster than map segmentation; however, it

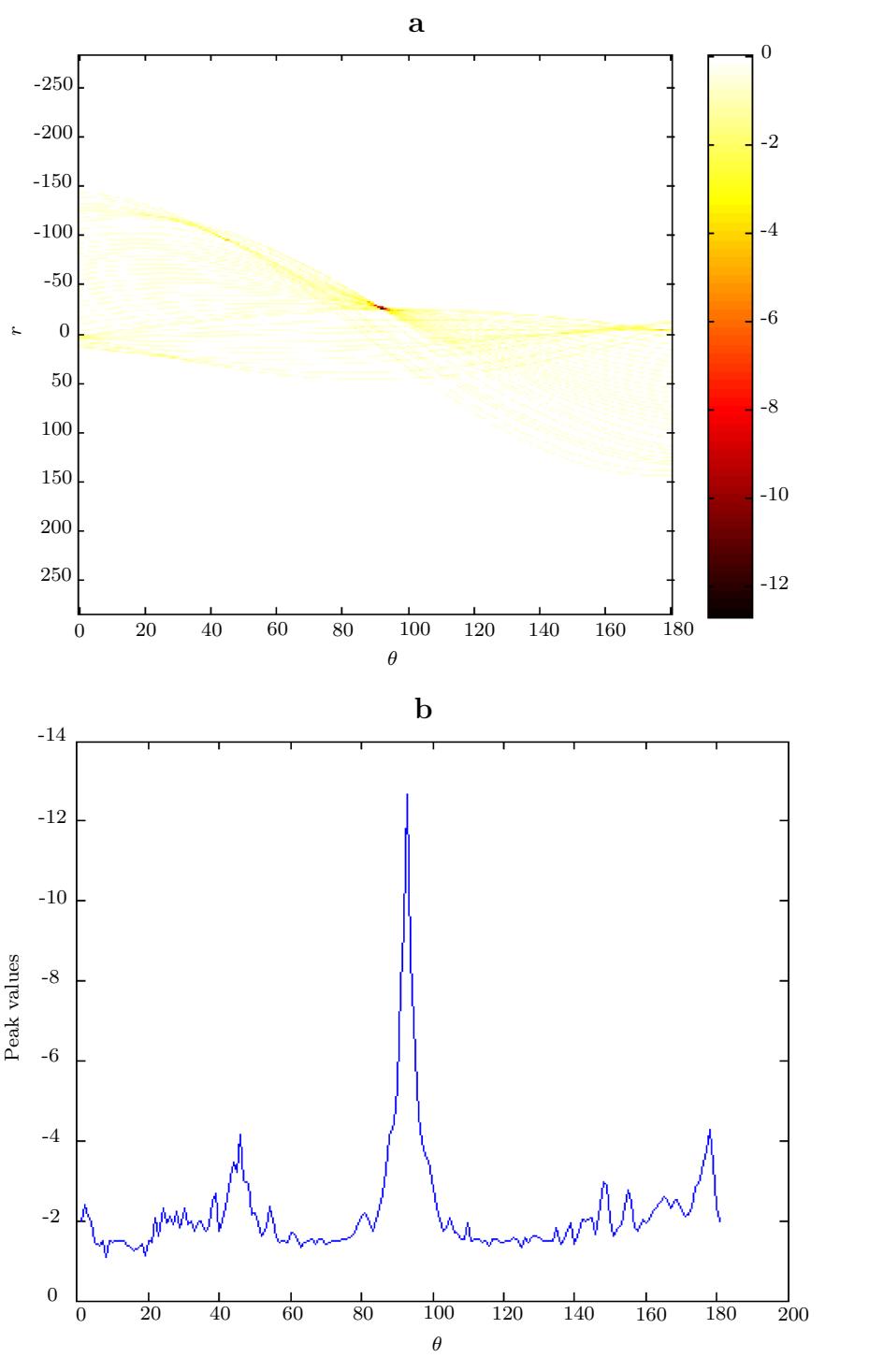


Figure 4.28: **a)** Radon image of m_2 over 180 degrees **b)** peak point of the Radon image

is less scalable with respect to the size of the map.

As in the previous experiment, each robot generates a local occupancy grid map of

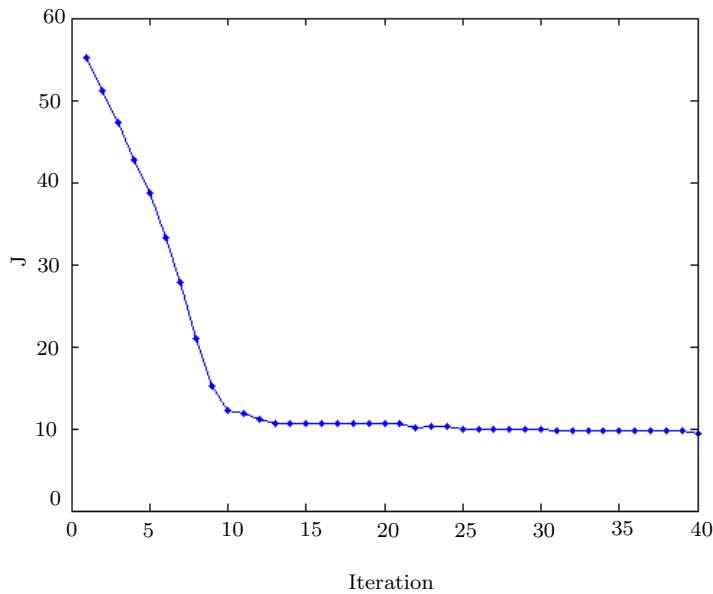


Figure 4.29: Performance index of the map fusion based on the squared distance error of the corresponding points. The error converges after 15 iterations to a fixed error value.

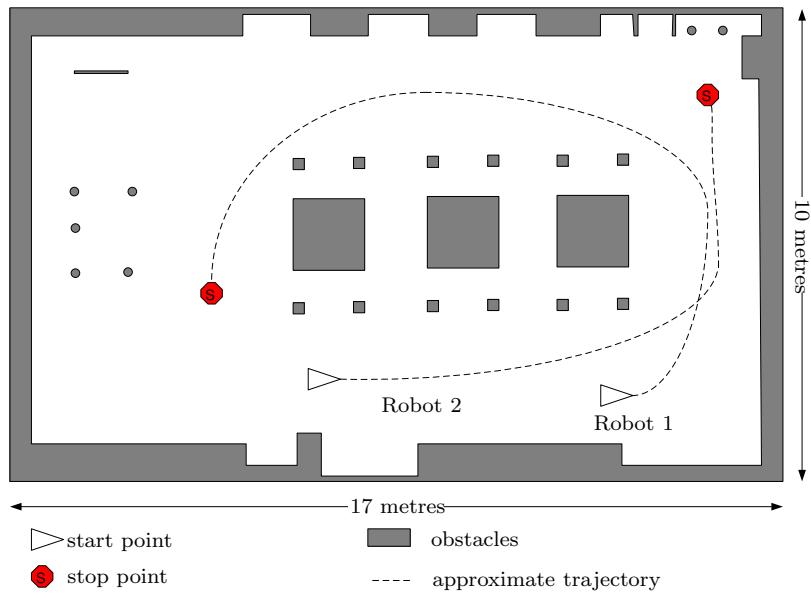


Figure 4.30: A more complex environment. Starting and ending locations of the robots are marked with triangles and stop signs, respectively. The robot trajectories are shown as dashed lines.

the environment. At certain specified time intervals, the robots share their local maps with each other. Fig. 4.31-a and Fig. 4.31-b show the two local maps generated by the robots. Note that the left part of the test environment was out of the range of the laser range scanners. The two maps, m_1 and m_2 , are provided by two robots. The proposed map fusion algorithm is assumed to be done on the first robot with m_2 being integrated into m_1 . Fig. 4.31-c shows the result of the map fusion of m_2 into m_1 using the proposed method. The transformation vector for this case is $trf = [x \ y \ \theta]^T = [-1.2 \ -32.9 \ 26^\circ]^T$. The verification index, $V(m_1, m_2)$ defined in equation (4.27) is 95%.

4.3.7.3 Discussion

The proposed method has been compared with three other methods: ARW map merging [2], map segmentation [121], and k-means clustering [123].

ARW and map segmentation were explained in Section 4.2. K-means clustering is an iterative partitioning where each point belongs to the cluster with the nearest mean. This method clusters points relatively quickly, but the clusters lack any kind of structured order. The structured ordering of the SOM neurons is the essential advantage that allows the collection of neurons to maintain the topological information contained in the map. Tamayo *et al.* state this as, “k-means clustering is a completely unstructured approach, which proceeds in an extremely local fashion and produces an unorganized collection of clusters that is not conducive to interpretation” ([124]). As a result, k-means is not a practical alternative to the SOM within the framework of the proposed algorithm in this work.

Table 4.4 summarizes the comparison of the processing times for the two experiments. In the first experiment, the algorithm takes about 13 seconds to run. This is about one sixth of the time that the algorithm presented in [121] required to run on the identical environment producing similar results. For the second experiment, the entire

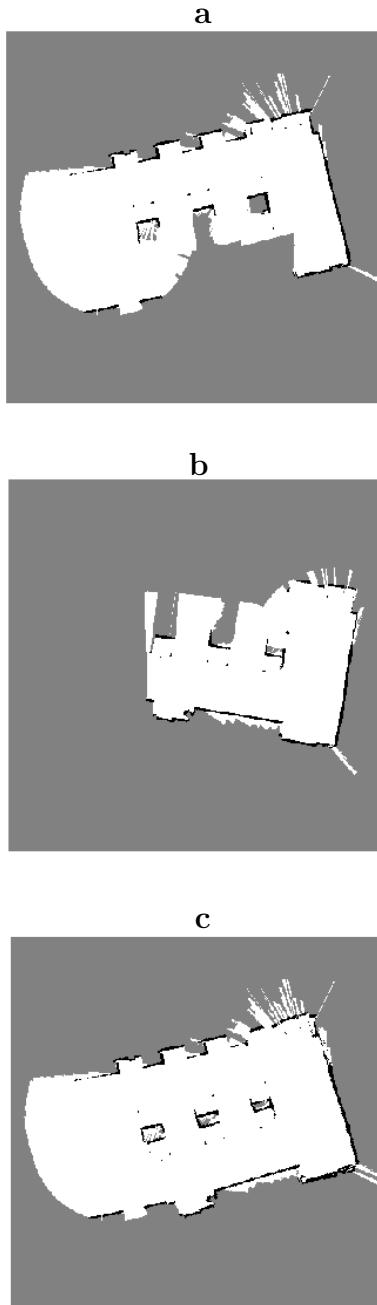


Figure 4.31: Local maps provided by the robots based on the approximate dimensions and trajectories of Fig. 4.30. **a)** m_1 provided by the first robot, **b)** m_2 provided by the second robot, **c)** both maps after the alignment by the proposed algorithm.

algorithm required about 16 seconds to run, which compares with over 1.5 minutes for the algorithm presented in [121]. To compare the results with ARW, a faster version of ARW is used where a set of 144 transformations have been investigated. In this case the proposed algorithm runs at least ten times faster. The reduced processing time is mainly due to the down-scaling of the map by SOM.

Table 4.4: Comparison of processing times

Method	Experiment 1	Experiment 2
SOM based	≈ 13 s	≈ 16 s
Map segmentation [121]	≈ 84 s	≈ 95 s
ARW map merging [2]	≈ 152 s	≈ 160 s

4.3.8 Summary

In this work, a new method for multiple-robot SLAM is developed. The proposed method uses a self-organizing map to reduce the complexity of the acquired occupancy grid maps. The result is that map fusion can be achieved more efficiently and reliably. Results are verified with tests performed in real environments. This is the first known application of the neural network theory to solve the multiple-robot SLAM problem. In summary, novel aspects of this approach include: preprocessing of occupancy grid maps using the map segmentation method, applying the self organizing map to learn and cluster the map, determining the relative rotation and the translation of two maps using the cluster points. A major advantage of this approach is its fast result due to the unsupervised training method of the SOM. This will allow faster exploration and mapping of unknown environments, which is useful in a variety of different robotics applications.

In the future, incorporating information gained if the robots do happen to see each other in the environment could yield better results because it provides a way for the robots to quickly and easily find their relative poses. Scalability of the proposed

algorithm to large numbers of robots should also be investigated further. Updating locations of the robots, once the relative transformations between them are known, will be described in Chapter 5.

4.4 Probabilistic Generalized Voronoi Diagram

In this section, a novel approach for occupancy grid map merging is proposed. In the general case, occupancy grid maps from multiple agents must be merged in real-time without any prior knowledge of their relative transformation. In addition, the probabilistic information of the maps must be accounted for and fused accordingly. In this work, the generalized Voronoi diagram (GVD) is extended to encapsulate the probabilistic information confined in the occupancy grid map. The new construct called the probabilistic GVD (PGVD) operates directly on occupancy grid maps and is used to determine the relative transformation between maps and fuse them. This approach has three major benefits over past methods: 1) It is effective for finding relative transformations quickly and reliably, 2) the uncertainty associated with transformations, used to fuse the maps, is accounted for, and 3) because of the probabilistic nature of the PGVD, parts of the maps that are more certain are preferentially used in the merging process.

4.4.1 Problems with Existing Methods and Motivations

Abstract geometrical perception is a foundation for high-level reasoning and knowledge sharing. When multiple robots are supposed to explore and map an unknown environment cooperatively, there is a need to provide a logical infrastructure so that robots can share their spatial perception and decide about how to use the shared knowledge. If the salient information from maps is extracted and shared among robots, the speed and accuracy of the mutual perception will improve and communi-

cation channels will not be burdened with large amounts of unprocessed data.

Retractions, also referred as skeletonizations or Voronoi graphs, are known to be efficient topological representations of complex maps. Voronoi graphs and graph matching in its different forms have many applications in SLAM.

There are many topological solutions for single-robot SLAM such as the works by Choset *et al.* [125], [126], annotated generalized Voronoi graph [127], the work by Beeson *et al.* [65], Bayesian inference [128] and semantic approach [129]; however, few solutions have been proposed for multiple-robot SLAM

In [126], a higher dimension generalized Voronoi graph (GVG) is proposed. The proposed GVG method is used to improve single-robot SLAM in [125]. Beeson *et al.* extend their work in [65], where a factored mapping framework based on the spatial semantic hierarchy is proposed. The factored mapping uses a symbolic skeleton representation to build large scale maps. However, this work is not extended to the multiple-robot map fusion. In [130], a solution to detect and recognize topological features in underground mines is proposed. In their solution, Delaunay triangulations on range scans are used to extract points of interest, such as intersecting corridors. Then the points of interest combined with a topological map is used for navigation. A probabilistic topological mapping using the Bayesian inference is presented in [128]. This work is not extended to the multiple-robot mapping.

Multi-robot SLAM based on topological map merging using both structural and geometrical characteristics of the Voronoi graph is proposed in [27]. The assumption in this work is that a robot will be able to recognize areas of the map that correspond to vertices. In this case the topological map is built on the occupied space as opposed to the free space. The method in [27] is claimed to be fast. However, a limitation is that the maps are not updated.

The proposed work in this section tries to solve three main problems:

- finding the relative transformation between maps,

- taking into account the uncertainty of the relative transformation, and
- updating the global map from the information in the local maps.

Solving these issues is necessary to have a better and faster map merging process.

In this work, a skeletonization approach is extended to be probabilistic and used for map matching of multiple robots to generate a global map. A common problem encountered when using retractions for SLAM is that the deformation retraction is difficult to find in a closed form and heuristic methods for generating it tend to be slow. In our approach, morphological operations are used to define the deformation retraction to generate the GVD as is explained in Section 4.4.3.2. The proposed approach is fast enough to be used in real-time and guarantees connectedness of the skeleton [66]. In addition, the probabilistic nature of the proposed PGVD allows areas of the maps that are known with higher certainty to be preferentially matched.

4.4.2 Definition

The generalized Voronoi diagram is a type of roadmap which is the locus of points equidistant to two or more obstacles. Before introducing the properties of the GVD, we will first present definitions of retraction, homotopy and deformation retraction.

Definition 4.4.1. *Retraction: A function that maps a manifold onto a subset of itself and has the property that any point in the subset is mapped to itself. Specifically, a retraction is any continuous function $f : X \rightarrow A$ with $A \subset X$ and $f(a) = a \forall a \in A$ [131].*

An intrinsic advantage of the retraction as a topological world representation is that it is invariant to translation and rotation. For multiple-robot SLAM, this capability can be used to fuse different maps by fusing corresponding retraction representations. The main benefit of this approach is that map matching will require less processing time since the abstract information from the maps is extracted and compared.

Definition 4.4.2. *Homotopy: A continuous function that smoothly deforms one function to another. Consider f and g as functions with domain U and range V . A homotopy is a continuous function $H : U \times [0, 1] \rightarrow V$ such that $H(x, 0) = f(x)$ and $H(x, 1) = g(x)$ [131].*

Definition 4.4.3. *Deformation Retraction: A homotopy that smoothly maps the identity function into a retraction. $H : X \times [0, 1] \rightarrow X$ with [131]*

$$\begin{aligned} H(x, 0) &= x, H(x, 1) \in A, \\ H(a, t) &= a \text{ with } a \in A \text{ and } t \in [0, 1] \end{aligned} \tag{4.35}$$

where $A \subset X$.

As mentioned, the GVD is the set of all points in the free space of a map that are equidistant from at least the two closest obstacles. The GVD is a retraction because all points on the GVD will be mapped to themselves and all points in free space will be mapped onto the GVD. Therefore, a function that smoothly maps the free space onto the GVD is a deformation retraction. This formulation leaves us with two important properties of the GVD that will be exploited:

1. The GVD is connected because the set of free space is connected and connectivity is maintained under a deformation retraction.
2. The GVD of a map is unique and is invariant to transformations and rotations because it is a retraction.

The GVD can be interpreted as a topological representation of the map structure that contains the key information intrinsic to the map but in a much more compact form.

There are different methods to generate GVD for a map. Mathematical morphological operations [66] are a fast and reliable method to build GVD.

In Blum's method introduced in [132], the GVD is developed using a maximal inscribed circle (MIC) method which is inefficient for large maps.

A dynamic version of the GVD is proposed by Boris *et al.* [133] which improves performance near non-convex obstacles. Though this method has good results, it is time consuming.

Beeson *et al.* [134] propose an extended Voronoi graph (EVG) algorithm to improve the efficiency of building the GVD when the robot is limited with its sensory horizon. Also, this method performs better by eliminating spurious junctions within noisy environments.

4.4.3 Description of the Method

In this section three main issues are investigated:

- transforming occupancy grid maps with an uncertain transformation,
- probabilistic map merging with the GVD, and
- map fusion with the entropy filter.

The first item is presented in the next section. The remaining two items are presented in the section after.

4.4.3.1 Transforming Occupancy Grid Maps with an Uncertain Transformation

Generally, any type of uncertainty in this context is identified as having five main causes: environment, sensors, robots, models and computations. Thrun *et al.* state that “Uncertainty arises if the robot lacks critical information for carrying out its task” [3].

Map fusion involves dealing with two types of uncertainty.

- *Uncertainty due to pose and sensor measurements.* This kind of uncertainty is represented within the occupancy grid map. Specifically, it originates from the uncertain pose of each robot and is embedded in the map of the robot. This can be considered as a form of uncertainty at the individual robot level. Throughout this research, the terms uncertainty or map uncertainty refer to this type of the uncertainty unless otherwise stated.
- *Uncertainty due to transformation:* If we can estimate the uncertainty in the transformation that relates two maps, then we can account for this uncertainty in the fusion process. In this work, the terms transformation uncertainty, rotation uncertainty and translation uncertainty are used to refer to this type of the uncertainty.

In this section, the second type of the uncertainty, transformation uncertainty, is studied. The result of this section is used in subsequent sections. To the authors knowledge, there is no known similar research in the literature.

The uncertainty associated with the relative transformation matrix is represented as a covariance matrix. The covariance error is propagated through the probabilistic transformation function using the linearized uncertainty propagation (LUP) formulation, a novel map merging method proposed in this work. This section addresses the propagation of the uncertainty in transformation given the transformation matrix and its covariance as a map is being transformed so that it may be merged with another. First the formulation for a single cell is explained, then it is extended to cover all cells.

Assume that $q = [q_i \ q_j]^T$ is a multivariate random variable denoting the position of the cell (i, j) from map m . The probability distribution function (PDF) of q is shown by $p(q)$ and it is assumed to be a delta function. Mean value of q is shown by $\mu_q = [\mu_{q_i} \ \mu_{q_j}]^T$. The point q is transformed to another point, r , with the PDF of

$p(r)$. The transformation is performed by $f(\cdot)$ defined as:

$$r = f(\psi, x, y, q) = \begin{bmatrix} \cos \psi & \sin \psi \\ -\sin \psi & \cos \psi \end{bmatrix} q + \begin{bmatrix} x \\ y \end{bmatrix}, \quad (4.36)$$

where ψ is the rotation angle and x, y are translation elements.

In general, the nonlinearity and uncertainty of the transformation will cause a non-Gaussian distribution for r . However, it is possible to approximate the non-Gaussian distribution with a Gaussian distribution. By linearizing the transformation using the first order Taylor expansion, a Gaussian distribution is assigned to the transformed point:

$$p(r) = |2\pi\Sigma_r|^{-\frac{1}{2}} \exp \left\{ \frac{-(r - \mu_r)^T \Sigma_r (r - \mu_r)}{2} \right\} \sim \mathcal{N}(r; \mu_r, \Sigma_r), \quad (4.37)$$

the mean of the normal distribution, μ_r is calculated by (4.36). To calculate the covariance, assume that the covariance of the transformation matrix is

$$\Sigma_{tr} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{x\psi}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 & \sigma_{y\psi}^2 \\ \sigma_{x\psi}^2 & \sigma_{y\psi}^2 & \sigma_{\psi\psi}^2 \end{bmatrix}. \quad (4.38)$$

Through linearization, the covariance of the transformed point, Σ_r , is calculated by

$$\Sigma_r = F_{xy\psi} \Sigma_{tr} F_{xy\psi}^T + F_q \Sigma_q F_q^T \quad (4.39)$$

where $F_{xy\psi}$ and F_q are the Jacobians of $f(\cdot)$ defined as

$$F_{xy\psi} = \frac{\partial f(\psi, x, y, q)}{\partial(\psi, x, y)}, \quad F_q = \frac{\partial f(\psi, x, y, q)}{\partial(q)}, \quad (4.40)$$

and Σ_q is the covariance of the point q which is zero. By applying this approach, Σ_r

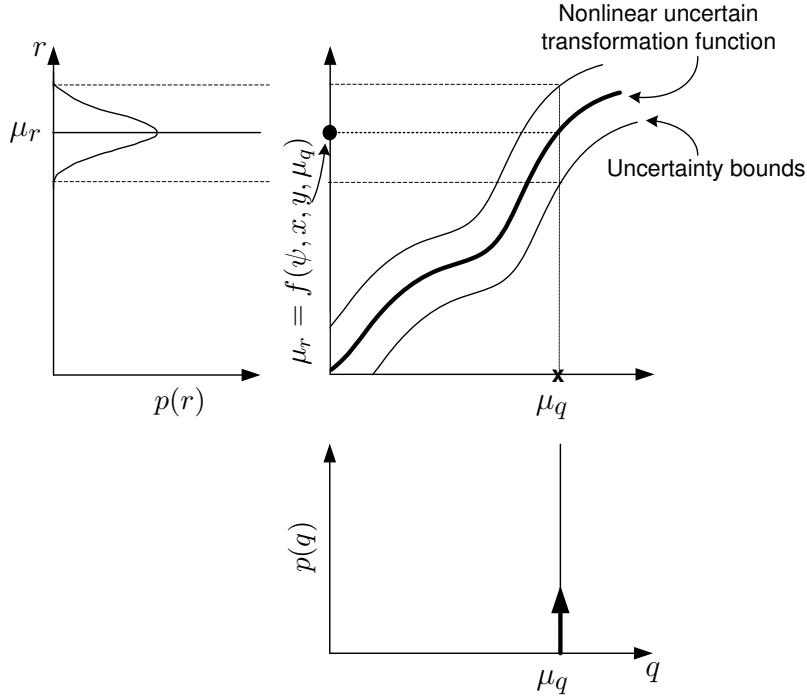


Figure 4.32: Lower right: point q with the distribution of $p(q)$ is transformed by an uncertain transformation function. Upper right: the uncertainty of the transformation function is shown by setting boundaries around the nominal function. Upper left: after linearization, the transformed point, r will have a Gaussian distribution, $p(r)$.

is calculated and shown in (4.41).

$$\Sigma_r = \begin{bmatrix} \sigma_{xx}^2 + 2\sigma_{x\psi}^2 r_1 + \sigma_{\psi\psi}^2 r_1^2 & \sigma_{xy}^2 + \sigma_{y\psi}^2 r_1 + \sigma_{x\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_1 r_2 \\ \sigma_{xy}^2 + \sigma_{y\psi}^2 r_1 + \sigma_{x\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_1 r_2 & \sigma_{yy}^2 + 2\sigma_{y\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_2^2 \end{bmatrix}, \quad (4.41)$$

where r_1 and r_2 are defined for the simplification of Σ_r as

$$\begin{aligned} r_1 &= -q_i \sin \psi + q_j \cos \psi \\ r_2 &= -q_i \cos \psi - q_j \sin \psi. \end{aligned} \quad (4.42)$$

Note that the covariance Σ_r is a function of q .

Fig. 4.32 depicts the problem with a hypothetical nonlinear function representing the transformation. The lower right shows the point q which should be transformed

by the given transformation function. The position of this point is deterministic, therefore its distribution is a delta function. However, the transformation function is not certain, shown by setting boundaries around the nominal transformation function (upper right). A Gaussian distribution is assigned to the transformed point (upper left).

Equation (4.39) gives the shape of the Gaussian distribution for every transformed point. Fig. 4.33 shows this process. The point q from map m is transformed to point μ_r . By applying the Gaussian defined in equation (4.39), depicted by $\mathcal{N}(r; \mu_r, \Sigma_r)$, the point μ_r takes a Gaussian form.

The next step is to extend this formulation to all points of the map. Algorithm 4.4.1 explains this process. For simplicity, we use $T_{x,y,\psi}(q)$ to denote that the point q is rotated according to ψ and then translated according to (x, y) . The same concept applies for transformation of a map, m , shown by $T_{x,y,\psi}(m)$.

First, the map m is transformed according to the given transformation, shown by $T_{x,y,\psi}(\cdot)$ (line 1). This is shown in Fig. 4.33, where the resulting map is marked by μ_m . In line 2, the final map which includes the uncertainty of the transformation is initialized by μ_m . In line 4, for every point of the transformed map, the covariance is calculated using equation (4.39). In line 5, the Gaussian kernel based on the covariance is calculated. Note that the Gaussian kernel is center originated which means the center point of the kernel is $H(0, 0)$. The mean of the kernel is the transformed point, while its 2×2 covariance matrix is calculated by equation (4.39). It is important to note that for each point of the transformed map, the kernel takes different values. Then in line 6 for every transformed point the Gaussian kernel is convolved with. The operator \circledast denotes the convolution of a Gaussian kernel with the map [66]

and defined as:

$$\begin{aligned} m'(k, l) &= \mu_m(k, l) \otimes H(k, l) \\ &= \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} \mu_m(i, j) H(k - i, l - j). \end{aligned} \quad (4.43)$$

For simplicity of implementation, it is assumed that the size of the kernel is the same as μ_m .

Now m' can be fused with its pair map using the entropy filter method detailed in Section 4.4.3.2.7.

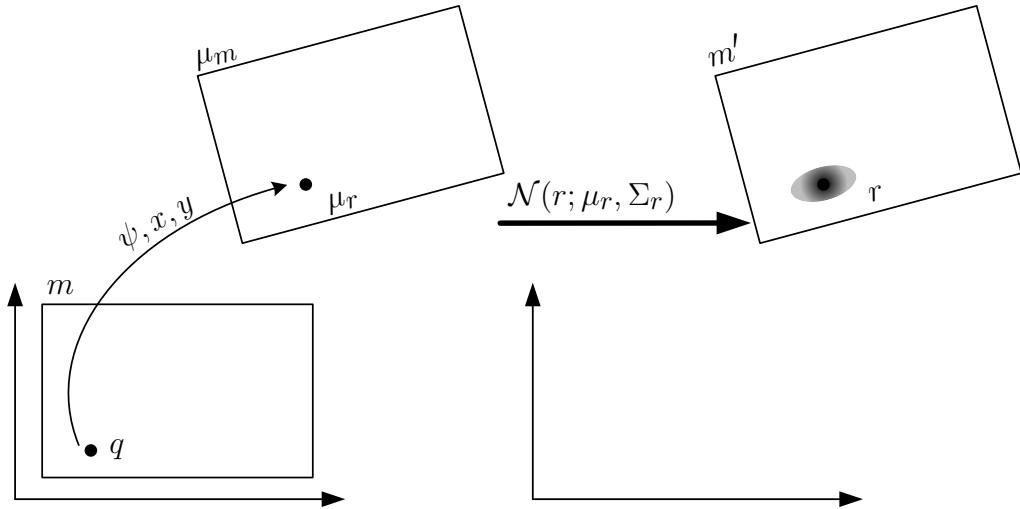


Figure 4.33: Linearized transformation uncertainty propagation. Point q is transformed to point μ_r according to the rotation ψ and the translation x, y . Then a Gaussian Kernel, $\mathcal{N}(r; \mu_r, \Sigma_r)$ is convolved with the transformed point. Σ_r is calculated through linearization.

4.4.3.2 Probabilistic Map Merging with the GVD

For simplicity, consider the case of two mobile robots, R_1 and R_2 , equipped with laser rangers exploring an environment and building occupancy grid maps (OGM) [3]. In an OGM representation, each cell in map $m_k(i, j)$ is a binary random variable (RV) where $p(m_k(i, j) = 1) = p(m_k(i, j))$ is the probability that the cell at location (i, j)

Algorithm 4.4.1 Linearized Uncertainty Propagation.

Input: Occupancy grid map: m ,
Transformation: ψ, x, y ,
Uncertainty of the transformation: Σ_{tr} .
Output: Transformed occupancy grid map: m' .

```
1:  $\mu_m = T_{x,y,\psi}(m)$ 
2:  $m' = \mu_m$ 
3: for all  $\mu_r = [\mu_{r_i} \mu_{r_j}]^T \in \mu_m$  do
4:   Calculate  $\Sigma_r$  using (4.39)
5:    $H \leftarrow \mathcal{N}(\mu_r, \Sigma_r)$ 
6:    $m'(\mu_{r_i}, \mu_{r_j}) \leftarrow \mu_m(\mu_{r_i}, \mu_{r_j}) \otimes H(\mu_{r_i}, \mu_{r_j})$ 
7: end for
```

is occupied. It is convenient to represent the values in the OGM using the log odds representation of occupancy [3]:

$$l_k(i, j) = \log \frac{p(m_k(i, j))}{1 - p(m_k(i, j))} \quad (4.44)$$

Without loss of generality, assume that R_2 transmits its local map, map_2 to R_1 through a wireless channel. R_1 is now responsible for incorporating the transmitted map into its own local map, map_1 . There are three main challenges that need to be overcome:

1. The relative transformation from map_1 to map_2 needs to be found.
2. The uncertainty of the transformation should be calculated and accounted for.
3. The OGM probabilities from map_2 need to be incorporated with the OGM probabilities of map_1 .

The relative transformation between the maps is a result of the relative initial poses of the robots. Assuming that there is no access to a global reference, the only way to extract this transformation is by processing the maps themselves. The proposed probabilistic skeleton method accounts for all issues.

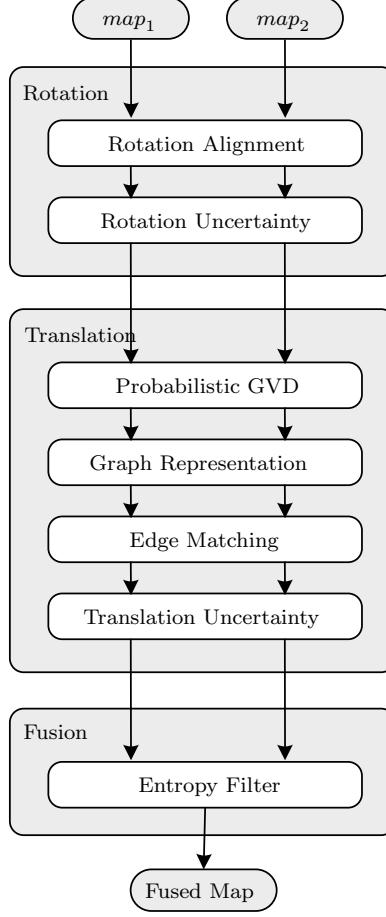


Figure 4.34: The proposed map fusion algorithm. Two input maps, map_1 and map_2 , are fused by finding their relative transformation matrix. No prior information is available regarding the relative position of the two robots.

An overview of the elements of the algorithm is shown in Fig. 4.34. In the first block, *Rotation Alignment*, the relative orientation between the maps is found using the Radon transform. The covariance of the orientation is then calculated. The rotated map is oriented given the rotation and its covariance. This is performed in the *Rotation Uncertainty* block using Algorithm 4.4.1 assuming zero translation. Then in the *Probabilistic GVD* block, the GVD is extracted while taking into account the map uncertainty of the OGM . The GVD in the *Graph Representation* block is formulated mathematically. Then in the *Edge Matching* block, edges of the graphs are cross matched to find the translation. Based on the matching edges, the covariance of the translation is calculated. In the *Translation Uncertainty* block, this covariance

is used to translate the map taking into account the uncertainty of the translation, using Algorithm 4.4.1 assuming zero rotation. Finally in the *Entropy Filter* block, the aligned maps are fused by an entropy filter. The subsequent sections will describe each of the block components in detail.

4.4.3.2.1 Rotation Alignment

In structured environments like urban or indoor settings, the relative rotation between maps can be found easily using the Radon transform as shown by the authors in previous work [121]. The Radon transform is the projection of the image intensity along a radial line oriented at a specific angle. Peak points in the Radon transform will correspond to straight line segments in the image. As a result, it is possible to resolve the relative rotation, ψ , between two images by looking for peaks in the Radon images of both maps.

$$\psi = \arg \min_{\theta} \mathcal{R}_{\theta=0:180}(map_1) - \arg \min_{\theta} \mathcal{R}_{\theta=0:180}(map_2), \quad (4.45)$$

where $\mathcal{R}_{\theta=0:180}(map)$ is the Radon image of the input map over the specified domain of angles (θ). Due to environment similarity, four rotation hypotheses are considered and only one is accepted by a similarity index [121].

At the output of this block there are the two aligned maps, m_1 and m_2 with the same size of $m \times n$, given by:

$$\begin{aligned} m_1 &= map_1 \\ m_2 &= T_{0,0,\psi}(map_2) \end{aligned} \quad (4.46)$$

where $T_{x,y,\psi}(m)$ transforms m by a translation (x, y) and a rotation ψ .

4.4.3.2.2 Rotation Uncertainty

The uncertainty in the relative rotation must be accounted for in the subsequent calculation of the relative translation. This is performed using Algorithm 4.4.1. This means that after finding the rotation, its uncertainty on the map is propagated using Algorithm 4.4.1, assuming zero translation and $\sigma_{xx}^2 = \sigma_{yy}^2 = \sigma_{xy}^2 = 0$. Now the process of finding the relative translation can be done knowing that uncertainty of the rotation is already incorporated into the rotated map.

4.4.3.2.3 Building the Probabilistic GVD

Finding the relative translation between maps is much more challenging than finding the relative rotation. The search for overlapping parts of maps can be slow. As a result, we use a novel probabilistic topological representation of the map as described in this section.

The probabilistic GVD is found for each of the two maps, m_1 and m_2 . In general this process is completed in two steps:

1. Find the GVD efficiently using mathematical morphological operations on the binary OGM.
2. Compute the associated probabilities of each cell in the GVD based on the actual probabilities in the OGM.

Mathematical morphology (MM) defines the application of set operations on binary images using convolution between the image and defined masks. It has been used extensively in computer vision and image processing. A set of basic operators are defined in MM like erosion, dilation, opening, closing, skeleton and hit-or-miss transform with different properties. The most important property is that they are translation invariant.

The GVD of the binary map is generated using eight D-type hit-and-miss transform masks [66]. Each mask is designed for a particular situation to guarantee the con-

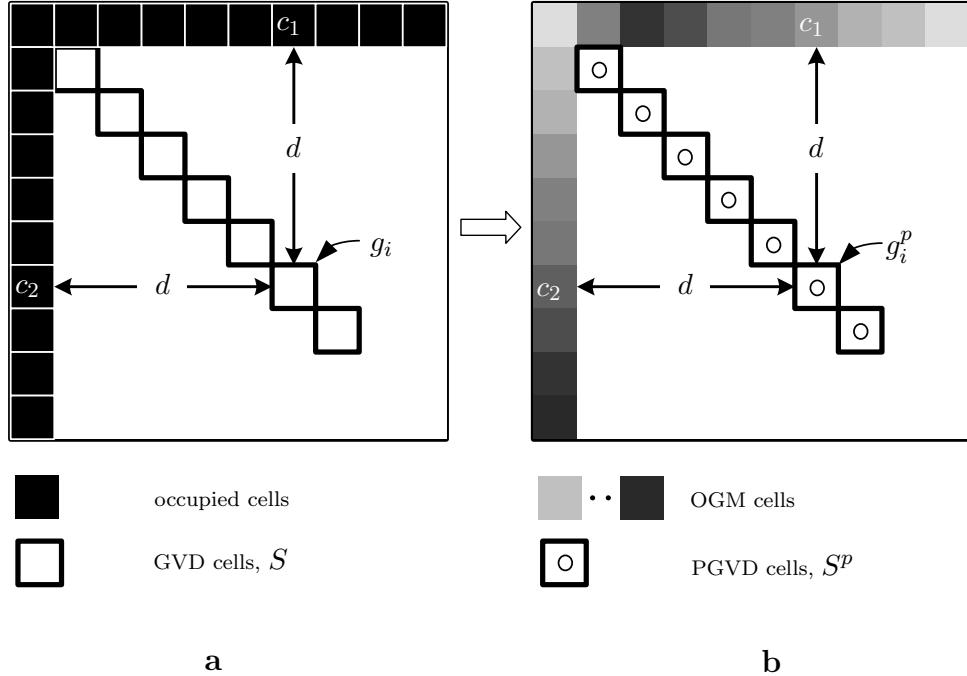


Figure 4.35: **a)** The GVD generated from the OGM. **b)** The PGVD contains probabilistic information based on the probabilities in the OGM

nectedness (using connectivity-8 model). The GVD is represented as a matrix, S , with the same size as the map, $(m \times n)$, defined as:

$$S = [s_{ij}]_{i=1..m, j=1..n} \quad (4.47)$$

$$s_{ij} = \begin{cases} 1 & \text{if } m(i, j) \in \text{GVD} \\ 0 & \text{otherwise} \end{cases} \quad (4.48)$$

Fig. 4.35-a depicts a simple environment with the occupied cells and the GVD, S . The skeleton developed by the proposed approach does reflect the uncertain nature of the OGM. While the GVD structure is generated from a binary version of the OGM, the probabilistic information in the OGM should be reflected in the GVD structure. A new structure, coined the probabilistic GVD (PGVD) is built that combines the structure of the GVD with the probabilistic nature of the OGM and is shown in Fig. 4.35-b.

Definition 4.4.4. *The Contact Points of a cell in the GVD are all the occupied cells in m that have the same distance from the cell as the closest occupied cell.*

For example in Fig. 4.35-a, c_1 and c_2 are contact points of g_i .

Definition 4.4.5. *Probabilistic GVD: Each cell in the GVD is represented by a binary RV representing the probability that it has two or more occupied contact points based on their probabilities of occupancy in the OGMs.*

Consider that a GVD, S , contains s cells, $g_i, i = 1..s$. Each cell in the GVD has an associated set of n_i contact points in the binary map m .

The PGVD, S^p has the same structure as the GVD, except that each of the s cells is represented as a binary RV, $g_i^p, i = 1..s$ where $p(g_i^p = 1) = p(g_i^p)$ is the probability that cell g_i^p has been correctly placed in the GVD. Each cell in the PGVD has an associated set of contact points $C_i = \{c_1, c_2, \dots, c_{n_i}\}, i = 1..s$, where these contact points are at the same locations as the contact points of g_i except they are in the OGM so they have associated probabilities of occupancy. Each contact point is a binary RV where $p(c_j = 1) = p(c_j) \forall j = 1..n_i$ is the probability that the contact point is occupied taken directly from the OGM.

A cell belongs in the GVD if at least two of the contact points are occupied. For each cell in the PGVD g_i^p , we must determine the probability that it has at least two occupied contact points based on their probabilities of occupancy.

To do so, another RV $n_{g_i^p}$ is defined which represents the probability distribution of the number of occupied contact points of g_i^p . $p(n_{g_i^p} = k)$ is the probability that cell g_i^p contains k occupied contact points.

The $p(n_{g_i^p} = k)$ is defined as a function of n_i , the number of contact points in the OGM, and k , the number of contact points that are actually occupied:

$$p(n_{g_i^p} = k) = f(n_i, k). \quad (4.49)$$

The function $f(n_i, k)$ can be defined recursively as:

$$f(n_i, k) = p(c_{n_i}) f(n_i - 1, k - 1) + (1 - p(c_{n_i})) f(n_i - 1, k). \quad (4.50)$$

Intuitively, this equation says that the probability of having k out of n_i contact points occupied is equal to the probability that contact point c_{n_i} is occupied and that $k - 1$ of the remaining $n_i - 1$ contact points are occupied, plus the probability that c_{n_i} is not occupied and k of the remaining $n_i - 1$ contact points are occupied.

The base cases for the recursion are cases where all of the cells must be occupied or none of the cells must be occupied:

$$\begin{aligned} f(n_i, n_i) &= \prod_{j=1}^{n_i} p(c_j) \\ f(n_i, 0) &= \prod_{j=1}^{n_i} (1 - p(c_j)) \end{aligned} \quad (4.51)$$

We will proceed to prove that this function $f(n_i, k)$ provides the valid probability density function for the RV $n_{g_i^p}$.

Theorem 4.4.1.

$$\sum_{k=0}^n f(n, k) = 1 \quad (4.52)$$

Proof. By induction:

Base case: $n = 1$

$$\sum_{k=0}^1 f(1, k) = f(1, 0) + f(1, 1) = p(c_1) + (1 - p(c_1)) = 1 \quad (4.53)$$

from equations (4.51) and (4.51).

Induction step:

Assume true for $n - 1$:

$$\sum_{k=0}^{n-1} f(n-1, k) = 1 \quad (4.54)$$

Consider n :

$$\begin{aligned} & \sum_{k=0}^n f(n, k) \\ = & f(n, 0) + f(n, 1) + \dots + f(n, n-1) + f(n, n) \\ = & -p(c_n) f(n-1, 0) + f(n-1, 0) \\ & + p(c_n) f(n-1, 0) - p(c_n) f(n-1, 1) + f(n-1, 1) \\ & + p(c_n) f(n-1, 1) - p(c_n) f(n-1, 2) + f(n-1, 2) \\ & + p(c_n) f(n-1, 2) - p(c_n) f(n-1, 3) + f(n-1, 3) \\ & + \dots \\ & + p(c_n) f(n-1, n-1) \\ = & f(n-1, 0) + f(n-1, 1) + \dots + f(n-1, n-1) \\ = & \sum_{k=0}^{n-1} f(n-1, k) = 1 \end{aligned}$$

Induction is complete. Therefore $f(n_i, k)$ is a valid probability density function that produces all possible combinations of contact points and sums to 1.

□

Corollary 4.4.1. *The number of additive terms in $\sum_{k=0}^{n_i} f(n_i, k)$ is 2^{n_i}*

Proof. The number of additive terms in $f(n_i, k)$ is

$$\binom{n_i}{k} = \frac{n_i!}{k!(n_i - k)!} \quad (4.55)$$

It is also well known from combinatorial analysis by summing the entries in Pascal's

triangle that:

$$\sum_{k=0}^{n_i} \binom{n_i}{k} = 2^{n_i} \quad (4.56)$$

□

$p(g_i^p)$ is now given by:

$$\begin{aligned} p(g_i^p) &= p(n_{g_i^p} \geq 2) = \sum_{k=2}^{n_1} p(c_i = k) \\ &= 1 - f(n_i, 0) - f(n_i, 1) \end{aligned} \quad (4.57)$$

The PGVD is now defined as:

$$S^p = \bigcup_{i=1}^s g_i^p \quad (4.58)$$

The output from this block will be a PGVD S_k^p for each input map m_k with $k = \{1, 2\}$.

This new probabilistic topological representation of the environment will be shown in subsequent sections to have significant advantages.

4.4.3.2.4 Graph Representation

The PGVD is then processed to be represented as a graph with edges and vertices.

A vertex can be identified as any cell in the PGVD with more than two adjacent occupied cells. Fig. 4.36-a shows a sample skeleton with vertices marked with ‘v’.

The set of all vertex points is defined as V .

To identify edges in the graph, a dilation mask is applied to each vertex in V . The dilation mask is shown in Fig. 4.36-b. The result of the dilation operation is a new map D (Fig. 4.36-b) that contains the dilated vertices. The edge matrix E is found as $S^p - D$, as shown in Fig. 4.36-c. This operation is performed on each of the two GVDs to produce two edge maps: $E_k, k = \{1, 2\}$.

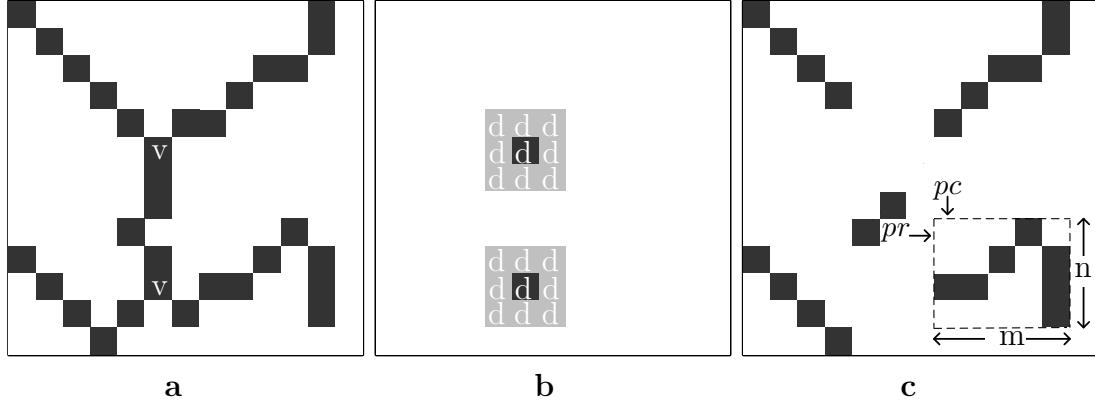


Figure 4.36: **a)** A sample GVD with vertices marked by a ‘v’ **b)** Vertices are dilated by morphological Dilatation operator to extract edges **c)** Edges of the GVD. Cells bounded by dashed lines depict the bounding matrix of an edge.

4.4.3.2.5 Edge Matching for Translation Alignment

The probabilistic edge matrices (PEM), E_1^p and E_2^p , have the same structure as the edge matrices, E_1 and E_2 , but the cells are the probabilistic ones extracted from the PGVD. These PEMs are now used to find the translational transformation between the two maps m_1 and m_2 . Edges with short lengths are removed to avoid processing short edges which have a higher chance of producing false positives. The edges of each probabilistic edge matrix are matched using a 2D cross correlation. To speed up the cross correlation, each edge from each edge matrix is represented as a sub-matrix of E where its size is such that it is as small as possible while still containing the entire edge. Fig. 4.36-c shows an edge bounded by a dashed boundary line.

For a given edge matrix, E_k^p with N_k edges, each edge, $e_k^{i_k}$, $i_k = 1..N_k$ is given by (refer to Fig. 4.36-c):

$$e_k^i = E_{k(pr_i:pr_i+m_i,pc_i:pc_i+n_i)}^p \quad (4.59)$$

To perform matching, each edge of E_1^p is cross correlated with each edge of E_2^p .

The 2D cross correlation of two matrices $e_1^{i_1}$ and $e_2^{i_2}$ with the respective sizes ($m_1 \times n_1$)

and $(m_2 \times n_2)$ is given as

$$\begin{aligned}\mathcal{E}^{(i_1, i_2)} &= [\epsilon_{i,j}]_{i=0..m_1+m_2-1, j=0..n_1+n_2-1} \\ &= e_1^{i_1} \star e_2^{i_2} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} e_1^{i_1}(k, l) e_2^{i_2}(k + i, l + j)\end{aligned}\quad (4.60)$$

where $\mathcal{E}^{(i_1, i_2)}$, the cross correlation matrix of edge $e_1^{i_1}$ and edge $e_2^{i_2}$, has size $(m_1 + m_2 - 1) \times (n_1 + n_2 - 1)$. The operator \star denotes the 2D cross correlation operation. The maximum value in the $\mathcal{E}^{(i_1, i_2)}$ matrix quantifies the best match between $e_1^{i_1}$ and $e_2^{i_2}$ based on the all possible combinations of translations between the two edge matrices. This value is computed using equation (4.60) for every combination of edges: $i_1 = 0..N_1$ and $i_2 = 0..N_2$.

Then we can define the similarity matrix, Γ_1^2 , between E_1^p and E_2^p as:

$$\begin{aligned}\Gamma_1^2 &= [\gamma_{i_1 i_2}]_{i_1=1..N_1; i_2=1..N_2}, \\ \gamma_{i_1 i_2} &= \max(\mathcal{E}^{(i_1, i_2)}).\end{aligned}\quad (4.61)$$

The best candidate for a match corresponds to the maximum value in the similarity matrix Γ_1^2 :

$$(i_1^*, i_2^*) = \operatorname{argmax}_{i_1, i_2} (\Gamma_1^2(i_1, i_2)), \quad (4.62)$$

where i_1^* and i_2^* are the indices of the most likely similar edges from the two maps.

The relative translation is calculated by resolving the translation vector which matched these two edges using the following relation [121]:

$$tr = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \mu_x(e_1^{i_1^*}) - \mu_x(e_2^{i_2^*}) \\ \mu_y(e_1^{i_1^*}) - \mu_y(e_2^{i_2^*}) \end{bmatrix} \quad (4.63)$$

where the $\mu_x(\cdot)$ and $\mu_y(\cdot)$ functions return the mean values of the elements of the

input matrix, evaluated along the X and Y axes respectively.

The final transformed map is defined as:

$$m'_2 = T_{x,y,0} (T_{0,0,\psi}(map_2)) = T_{x,y,\psi}(map_2) \quad (4.64)$$

It should be emphasized that the associated probabilities for each cell in the edges of the PGVD play an important role in determining the values from the cross-correlation. Edges with cells with higher probability of being true GVD cells will result in higher cross-correlation values.

False matches can be identified by analyzing the expected results of the 2D cross correlation.

Assume that there is an edge with l_1 cells within $e_1^{i_1^*}$ and an edge with l_2 cells within $e_2^{i_2^*}$. Assume that the best match in the 2D cross correlation results in L matching cells. Define the subset of cells in $e_k^{i_k}$ that were matched as $\varepsilon_k^{i_k}$. We can define the average confidence for the matched edge according to:

$$\alpha_k^{i_k} = \frac{1}{L} \sum_{l=1..L} p(\varepsilon_k^{i_k}) \quad (4.65)$$

By expanding equation (4.60), $\max(\mathcal{E})$ becomes $\alpha_1^{i_1} \alpha_1^{i_2} L$.

Given two edges with lengths l_1 and l_2 with unknown mutual matching cells, the best matching or maximum value in the correlation matrix should satisfy the following to be considered as a candidate match:

$$\max(\mathcal{E}^{(i_1, i_2)}) > \rho \alpha_1^{i_1} \alpha_1^{i_2} \min(l_1, l_2), \quad (4.66)$$

where ρ is a desired matching percentage. If there is no element in the Γ_1^2 matrix to satisfy equation (4.66), then there is no matching between the two maps.

4.4.3.2.6 Translation Uncertainty

The transformed map which included the effect of the rotation uncertainty was used to find the translation. After finding the translation, Algorithm 4.4.1 is used again to propagate the uncertainty of the translation, assuming zero rotation and $\sigma_{\psi\psi}^2 = 0$. We now have a fully probabilistic representation of the required transformation and can proceed to fuse the maps.

4.4.3.2.7 Map Fusion with the Entropy Filter

After finding the relative transformation between the two maps, the probabilities are combined and filtered to produce the final map. The data that is received in m_2 is akin to a batch of sensor data and should be incorporated by using the additive property of the log odds representation of occupancy originally defined in equation (4.44).

$$l_{fused}(i, j) = l_1(i, j) + l_2(i, j) \quad (4.67)$$

for all $i = 1..n, j = 1..m$. The probabilities can then be recovered from equation (4.44).

The probabilities can then be recovered from the log odds representation using:

$$p(m_{fused}(i, j)) = 1 - \frac{1}{\exp \{l_{fused}(i, j)\}} \quad (4.68)$$

The entropy filter is applied to the fused map, m_{fused} [135]. The entropy filter compares the original map, m_1 and the fused map m_{fused} and rejects updates that result in higher entropy.

For the case of a discrete binary RV, such as each cell of the OGM, $m(i, j)$ with $p(m(i, j)) = p_{ij}$, the entropy can be described by [58]:

$$H(m(i, j)) = -p_{ij} \log p_{ij} - (1 - p_{ij}) \log (1 - p_{ij}) \quad (4.69)$$

Mutual information, I_{ij} is defined as the reduction in entropy at location (i, j) between the original map, m_1 and the fused map, m_{fused} :

$$I_{ij} = H(m_1(i, j)) - H(m_{fused}(i, j)) \quad (4.70)$$

The final map, m_{final} is defined as:

$$m_{final}(i, j) = \begin{cases} m_{fused}(i, j) & I_{ij} \geq 0 \\ m_1(i, j) & I_{ij} < 0 \end{cases} \quad (4.71)$$

where only values from m_{fused} which resulted in positive information are kept. It should be noted that maps that are fused are always the original maps which contain no information from others. This is to avoid having overconfidence by fusing maps repeatedly.

4.4.4 Advantages and Disadvantages

To summarize, the proposed novel method for map fusion has a few key advantages:

- It accounts for uncertainties in the occupancy grid maps, using the probabilistic GVD,
- It considers the uncertainty of the calculated transformation by linearizing the transformation,
- It is fast and robust compared to other methods,
- It is able to preferentially match areas of the maps that are more certain.

It should be mentioned that the assumption in this work is that individual maps developed by each robot is accurate and robots can generate consistent maps.

4.4.5 Contribution

The contribution of this research is a novel map fusion algorithm that exploits the properties of the GVD to achieve fast and accurate map fusion for large maps. In addition, the uncertainty in the maps is used to build a PGVD that encapsulates not only the topological structure of the map, but also the confidences associated with different areas of the map. Once the PGVDs are built, edges are matched using a 2D cross correlation that will preferentially match the areas of the PGVD that have higher confidences. The resulting transformation can align maps to generate a global map. However, there is an uncertainty associated with this calculated transformation which should also be propagated to the maps. The novel LUP approach proposed in this research accounts for the uncertainty of the transformation. In LUP, the transformation for each cell of the map is linearized, then the Gaussian uncertainty of the transformation is propagated to the transformed cell. This process is performed on all transformed cells, therefore the resulting map carries the uncertainty of the transformation. Map fusion is then achieved with an entropy filter. The result is a fast, reliable, and robust method of fusing maps for multiple-robot SLAM.

4.4.6 Experiment

To demonstrate the effectiveness of the proposed method, three experiments are presented. The first one is performed on a well known data set for multiple-robot SLAM. In this experiment all the details of the proposed algorithm are explained. The second experiment is another real world experiment performed with two CoroBot robots in an indoor environment in the basement of the University of New Brunswick (UNB). Finally, the last experiment shows the map merging with three maps of the department of electrical and computer engineering of UNB.

4.4.6.1 Case1: Radish Dataset

This experiment is performed on the robotics data set repository (Radish) Fort AP Hill data set [136]. This is an open source and well-known data set which is used as a benchmark data set for view-based multiple-robot SLAM. The raw laser ranger and encoder data are recorded in standard Player [137] format and fused by particle filtering [28].

Fig. 4.37-a and Fig. 4.37-b show two input maps before alignment, map_1 and map_2 . Fig. 4.37-c shows m_2 , map_2 after applying alignment method by Radon transform. The alignment is 5.5° . Fig. 4.37-d and Fig. 4.37-e show the GVDs of m_1 and m_2 . Fig. 4.37-f and Fig. 4.37-g show filtered PEMs of m_1 and m_2 . The Γ_1^2 matrix is shown in (4.74). From (4.62), $i^* = 4$ and $j^* = 3$ which means the fourth edge of the first map has the highest similarity to the third edge of the second map. These two edges are used to calculate the translation vector from (4.63). The final translation vector is $tr = [-25, -1]^T$ and the rotation is $\psi = 5.5^\circ$.

From (4.65), $\alpha_1^4 = 0.86$ and $\alpha_2^3 = 0.89$ are the average confidences of the matched edges. The lengths of these two edges were 44 and 45 cells. The matching percentage, ρ which represents matching confidence, was chosen to be 95%. According to (4.66)

$$(0.95)(0.86)(0.89) \min(44, 45) = 31.99 < 32.9, \quad (4.72)$$

therefore the match is valid. Note that by choosing $\rho = 95\%$, the match of e_1^1 with e_2^1 , represented in $\Gamma_1^2(1, 1)$ is disqualified because the associated average confidences of those edges were higher: $\alpha_1^1 = 0.91$, $\alpha_2^1 = 0.92$ and the lengths of the edges are 56 and 62.

$$(0.95)(0.91)(0.92) \min(56, 62) = 44.54 > 31.6. \quad (4.73)$$

However, by reducing the value of ρ to less values such as 70%, then this match will be accepted. In the case that there are multiple matches in Γ_1^2 , for each match

the translation is calculated and the average of all translations is used as the final translation.

$$\Gamma_1^2 = \begin{bmatrix} 31.6 & 2.0 & 6.4 & 6.1 & 1.9 & 0.8 & 5.4 \\ 12.9 & 1.9 & 2.0 & 11.7 & 0.9 & 0.5 & 1.6 \\ 11.9 & 2.8 & 5.4 & 3.0 & 1.4 & 0.5 & 2.5 \\ 10.7 & 6.7 & \textbf{32.9} & 1.0 & 5.3 & 1.8 & 8.1 \\ 9.1 & 1.9 & 8.9 & 2.3 & 0.9 & 0.5 & 1.6 \\ 5.9 & 0.4 & 2.1 & 0.4 & 0.4 & 0.2 & 1.9 \\ 3.2 & 9.9 & 3.9 & 0.5 & 11.6 & 4.0 & 6.9 \\ 0.9 & 7.2 & 2.9 & 0.3 & 10.2 & 3.5 & 4.1 \end{bmatrix} \quad (4.74)$$

To show the benefit of the probabilistic approach, Γ_{det} , the deterministic Γ , is shown in equation (4.75). In this case, matching is performed directly on the GVD instead of the PGVD. Edges that were on the periphery of the explored area in the GVD are now matched quite well, when, in reality, those edges do not represent well the structure of the discovered map. As a result, the correct match is much less clear. By comparing those elements from Γ_{det} shown by bold font, with the same elements in Γ_1^2 , it is obvious that how the probabilistic structure can eliminate false or weak matches. Fig. 4.37-h shows the final aligned maps.

$$\Gamma_{det} = \begin{bmatrix} 38 & 2 & 9 & 10 & 2 & 2 & 9 \\ 17 & 2 & 2 & 30 & 2 & 1 & 2 \\ 16 & 3 & 6 & 5 & 3 & 2 & 3 \\ 11 & 9 & \textbf{40} & 1 & 10 & 8 & 12 \\ 14 & 2 & 10 & 3 & 2 & 1 & 2 \\ 14 & 1 & 7 & 1 & 1 & 1 & 8 \\ 6 & 29 & 9 & 1 & 29 & 29 & 25 \\ 3 & 29 & 9 & 1 & 32 & 32 & 21 \end{bmatrix} \quad (4.75)$$

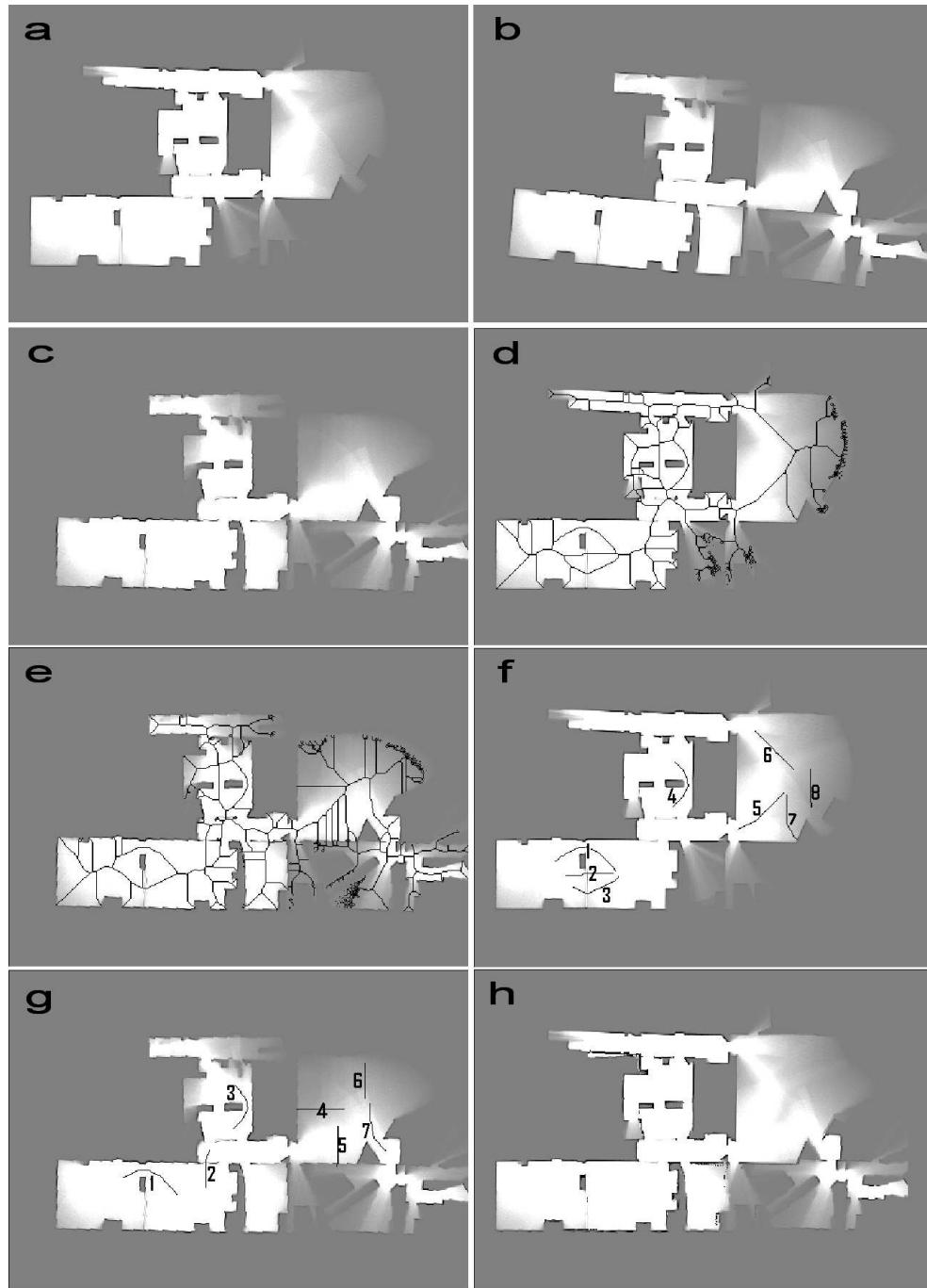


Figure 4.37: **a)** map_1 , **b)** map_2 , **c)** m_2 , rotated map_2 , **d)** PGVD of map_1 , **e)** PGVD of m_2 , **f)** PEM of map_1 , **g)** PEM of m_2 , **h)** Fused map.

4.4.6.2 Case2: Two CoroBot Robots

A real world experiment is performed with two CoroBots robots in an indoor environment in the basement of Head Hall, UNB. The total area of the environment is

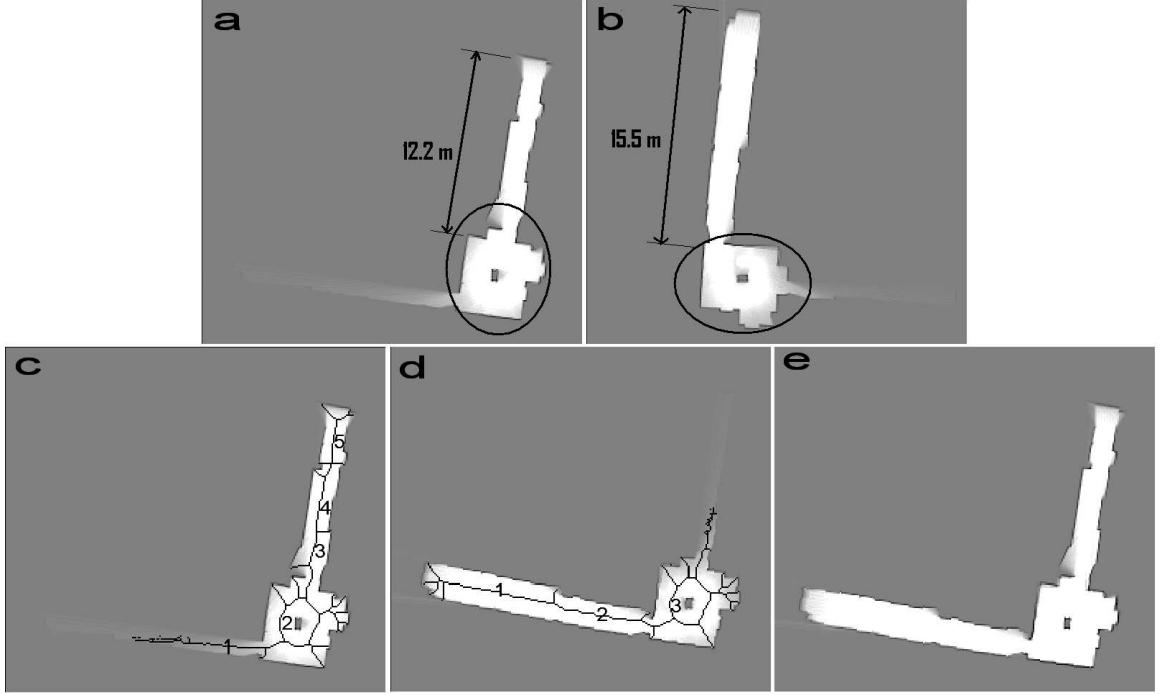


Figure 4.38: **a)** map_1 , **b)** map_2 , **c)**, PGVD and PEM of map_1 **d)**, PGVD and PEM of rotated map_2 **e)** Fused maps.

81.02 m^2 and the lengths of robot trajectories are 22.5 m and 25.9 m.

Fig. 4.38-a and Fig. 4.38-b show the OGMs built by the robots. The area of this experiment is not very large, but it is important because the overlap between two maps is approximately 25.4 m^2 which is 31% of the total area as shown by the ellipses. This shows the effectiveness of the algorithm with small overlaps. In both maps, there are non-overlapping corridors with almost the same size (15.5 m and 12.2 m). But the algorithm is capable of rejecting them as matching and finding the transformation based on the overlap. Fig. 4.38-c and Fig. 4.38-d depict the PGVD of the aligned maps with selected edges for matching marked with numbers. The edges marked with number 2 in Fig. 4.38-c and number 3 in Fig. 4.38-d are used to calculate the translation. Finally, Fig. 4.38-e shows the final fused map applying the proposed method.

4.4.6.3 Case3: Three CoroBot Robots

This experiment is performed in a larger environment, with an area of approximately 600 m^2 and three robots are involved. The trajectories of the robots are approximately 60 m, 35 m, and 55 m.

Fig. 4.49-a, b and c show the three local maps. Maps of Fig. 4.49-b and c are fused to Fig. 4.49-a. The overlapping region is indicated in the figure. The fused global map of the environment is shown in Fig. 4.49-d. By merging the maps, loop closure happens successfully.

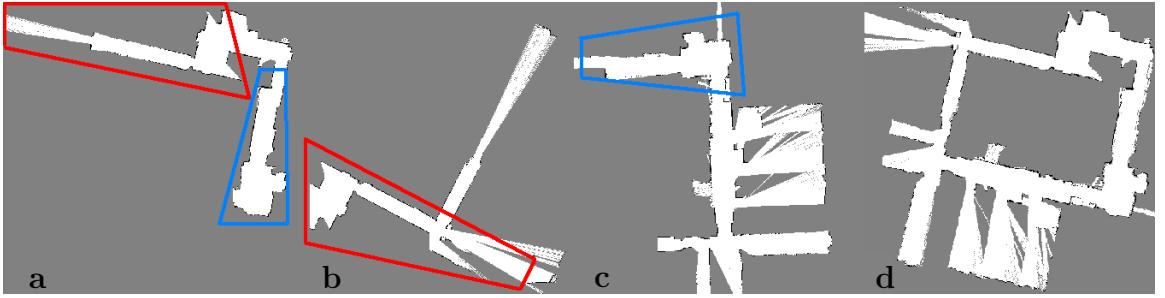


Figure 4.39: Three partial maps are fused together to generate a global map. (a) is the base map where (b) and (c) are fused to that. Overlaps are marked with polygons. (d) Final fused map which depicts loop closure.

4.4.6.4 Discussion: Comparisons

As mentioned, a major benefit of edge matching for map fusion is the low processing time requirement. The proposed method is compared with ARW map merging [2] and map segmentation [121], which were explained in Section 4.2.

Table 4.5 summarizes the comparison of the processing time and verification for all three experiments. As the results show, the proposed method operates at least eight times faster and the verification index [2] shows the accuracy of the results.

Table 4.5: Processing time and efficiency of three experiments, ① Radish data set, ② two CoroBots, ③ three CoroBots.

Experiment	(1)	(2)	(3)	(1)	(2)	(3)
Method	Processing (s)			Verification (%)		
PGVD based	12	10	13	95	91	92
Map segmentation [121]	105	83	106	95	92	94
ARW map merging [2]	168	150	152	93	88	92

4.4.7 Summary

A probabilistic multiple-robot SLAM algorithm has been presented that is fast and robust. The probabilistic generalized Voronoi diagram is used to fuse maps and account for uncertainties in the occupancy grid map. The proposed method has been shown to preferentially fuse areas of the maps that have low uncertainty. Experiments and comparison with other established methods show that it is effective and 9 to 14 times faster.

In future work, it would be desirable to use the structure of the GVD to verify the accuracy of the map matching. If different pairs of edges are matched with high accuracy, then the structure of the GVD could allow us to determine that both sets of edges correspond to the same transformation, increasing the likelihood of correct matching.

4.5 Hough Peak Matching

In this section, the mapping process is extended to multiple robots with a novel occupancy grid map fusion algorithm. Map fusion is achieved by transforming individual maps into the Hough space where they are represented in an abstract form. Properties of the Hough transform are used to find the common regions in the maps, which are then used to calculate the unknown transformation between the maps.

Finding the transformation is done in two steps. First, the relative rotation is calcu-

lated using map overlaps and the correlation of a function calculated over the Hough image. Then the translation is calculated by finding the overlaps of the maps in the Hough space. Results are improved through tuning and are finally verified by a similarity index.

Results are shown from tests performed on benchmark data sets and real-world experiments with multiple robotic platforms.

4.5.1 Problems with Existing Methods and Motivations

Performing map merging requires finding overlaps between maps and this usually requires a search operation. Generally this is a difficult task due to:

- **Modelling:** A mathematical representation for objects in maps is necessary to compare objects.
- **Partial views:** Map features from different view angles may look different.
- **Geometric relations:** The internal relation of map features is an inherent property that makes each map unique.

Performing this search in the Euclidean space is difficult. However, by transforming the map into a different representation, certain map properties can be exploited to make the problem easier. The Hough transform turns out to have many such properties. For example, line segments, which are common in most structured environments, are modelled as an intensity point in the Hough space. Relative relations of line segments are represented by the distances between intensity points. Additionally, since the Hough transform is generated over an angle range of 360° , then all views in the Hough space are included and the problem of partial views can be eliminated.

There are a variety of methods to extract geometric information like line segments from maps or laser ranger measurements [138, 139]. The Hough transform [66] and the Radon transform [115] are two classical methods for line extraction which have

different applications. The results of the Hough and the Radon transforms on a map are almost identical, although the approach of these two methods are different. The Radon transform is the projection of the image intensity along a radial line oriented at a specific angle, while the Hough transform is a parametric model of each non-background point. Other methods like Split-Merge are fast and effective for line extracting, but because of interesting properties of the Hough transform, in this work we will primarily use the Hough transform. The details of this construct are presented here.

As will be described, the Hough transform is an alternate way to represent the geometrical data within the map that has some useful advantages. This approach has been used for map merging in the past [31] where the rotation and translation are derived in separate steps similar to our approach. However, as is shown in the experimental results section, the main drawback of their solution is that it fails when there is not enough overlap between the input maps. Specifically, their approach to finding the relative translation depends on a map projection. If there is insufficient shared regions in projections, the proposed translation finding method produces wrong results. Our experimental results show that our method improves upon past methods in this respect.

4.5.2 Background: Hough Transform

The basic form of the Hough transform, as shown in Fig 4.40, maps every (x, y) point from the image space into the (θ, ρ) space, known as the Hough space, based on the following relation:

$$\rho = x \cos\theta + y \sin\theta. \quad (4.76)$$

As a result, each (x, y) point is represented as a sinusoid in the Hough space. The sine waves of a set of points belonging to a line segment all intersect at a point in the

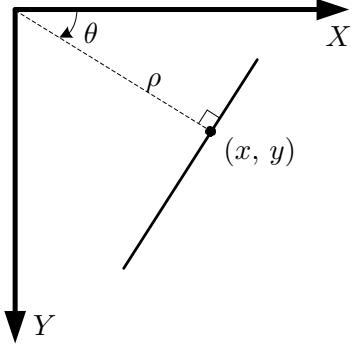


Figure 4.40: Hough modelling.

Hough space. Line segments in the image space with different lengths are mapped to points with different intensities proportional to the length of the lines. The image generated from the Hough transform of all points of the map is called the Hough image. Since the Hough image is periodic, it is sufficient to consider only a limited range of angles: $\theta \in [-\pi/2, \pi/2)$. Three useful properties of the Hough image are described here:

Property 4.5.1. *If a map, m is transformed to m' by a rotation ψ and a translation vector $tr = [\delta_x \ \delta_y]^T$, then the coordinates of the Hough space of the corresponding Hough images, $\mathcal{H}(\theta, \rho)$ and $\mathcal{H}'(\theta', \rho')$ will have the following relation [140] (the superscript T denotes the matrix transpose):*

$$\theta' = \theta + \psi \quad (4.77)$$

$$\rho' = \rho + \begin{bmatrix} \cos(\theta + \psi) & \sin(\theta + \psi) \end{bmatrix} tr, \quad (4.78)$$

Fig 4.41 shows this property of Hough image.

Property 4.5.2. *If $m = m_1 + m_2$ then the superposition relation applies to Hough images: $\mathcal{H}_m(\theta, \rho) = \mathcal{H}_{m_1}(\theta, \rho) + \mathcal{H}_{m_2}(\theta, \rho)$.*

Property 4.5.3. *A line segment with the length of L given by $y = a x + b$ in Cartesian*

coordinates, results in a point with intensity of L in the Hough space located at the point (θ, ρ) given by:

$$\theta = \arctan a - 90^\circ \quad (4.79)$$

$$\rho = b \sin \theta \quad (4.80)$$

$$\mathcal{H}(\theta, \rho) = L. \quad (4.81)$$

4.5.3 Description of the Method

Fig. 4.42 shows the algorithm for two robots with unknown relative positions and each with its own local map. Note that this algorithm is scalable and can be used for more than two robots. Maps in this algorithm are assumed to be in the form of occupancy grid maps.

According to Fig. 4.42, the flow diagram of the system, inputs are map_1 and map_2 . First, in the *Hough Rotation* block, a function is applied to the Hough images of the inputs to find the relative orientation. This block produces one approximate solution. In the *Multiple Hypothesis* block, multiple accurate estimates are produced. Results of these two block are combined to generate one accurate solution for rotation. Then this result is tuned in the *Tuning Rotation* block. After finding the orientation, in the *Hough Translation* block, an approximate translation is found using oriented Hough images. The approximate translation is adjusted in the *Entropy Tuning* block using image entropy. Finally, in the *Similarity Index* block, results are verified to ensure the accuracy of the map fusion.

For two given maps, m_1 and m_2 , assume that m_1 is composed of two parts, a set of cells, m , that exists in m_1 and m_2 and a set of cells Δ_1 that exist only in m_1 (Fig. 4.43). The same assumption applies to m_2 where m exists in both maps and Δ_2 exists only in m_2 :

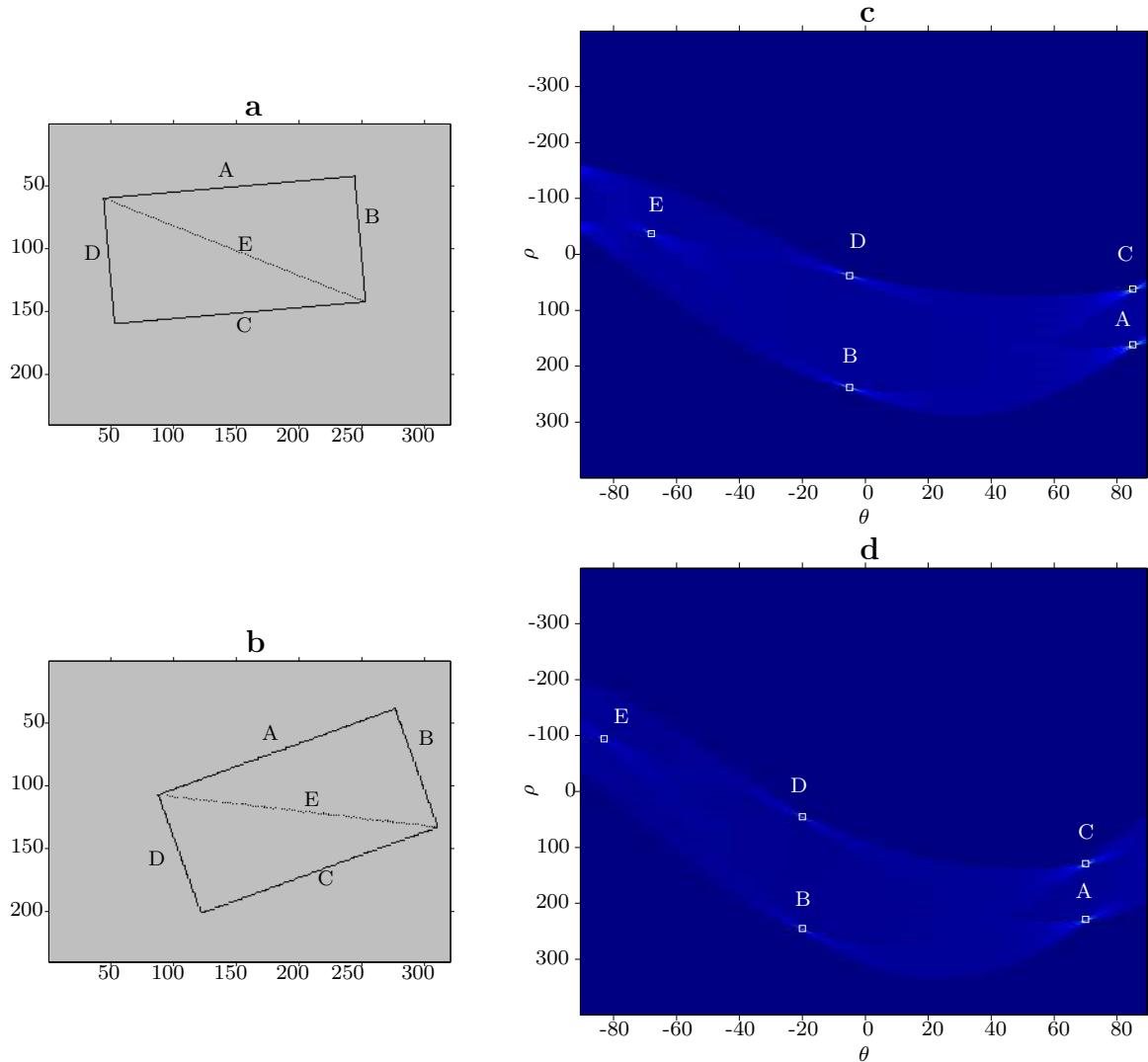


Figure 4.41: Effect of the transformation of maps on the Hough images according to Property 4.5.1 **a)** An example occupancy grid map. **b)** Hough image of the map over 180 degrees. Labeled points correspond to the the line segments. **c)** The map is rotated 15° and translated 50 cells along both axes. **d)** Hough image of the transformed map over 180 degrees

$$m_1 \triangleq \{m\} \cup \{\Delta_1\} \quad (4.82)$$

$$m_2 \triangleq \{m\} \cup \{\Delta_2\}. \quad (4.83)$$

According to Property 4.5.2, the Hough image of m_1 is:

$$\mathcal{H}_{m_1}(\theta, \rho) = \mathcal{H}_m(\theta, \rho) + \mathcal{H}_{\Delta_1}(\theta, \rho). \quad (4.84)$$

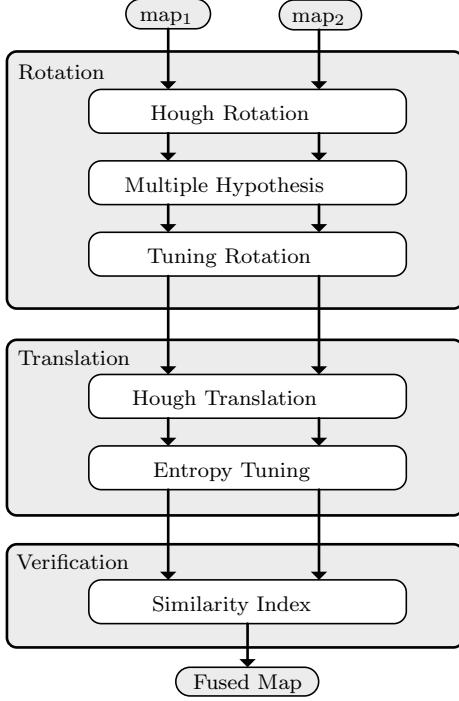


Figure 4.42: The proposed map fusion algorithm. The two input maps are fused by finding their relative transformation matrix. No prior information is available regarding the relative position of two robots.

Assume that ψ and $tr = [\delta_x \ \delta_y]^T$ are transformation elements that can fuse these two maps. According to Property 4.5.1 and Property 4.5.2, the Hough image of m'_2 , the transformed m_2 , becomes:

$$\begin{aligned} \mathcal{H}_{m'_2}(\theta, \rho) &= \mathcal{H}_m(\theta + \psi, \rho + \begin{bmatrix} \cos(\theta + \psi) & \sin(\theta + \psi) \end{bmatrix} tr) + \\ &\quad \mathcal{H}_{\Delta_2}(\theta + \psi, \rho + \begin{bmatrix} \cos(\theta + \psi) & \sin(\theta + \psi) \end{bmatrix} tr). \end{aligned} \quad (4.85)$$

If there is a method to identify the overlaps between maps, $\{m\}$, then the transformation is calculated by the fact that the Hough transform of $\{m\}$ from m_1 should be the same as the the Hough transform of $\{m\}$ from the transformed m_2 , m'_2 :

$$\mathcal{H}_m(\theta, \rho) = \mathcal{H}_m(\theta + \psi, \rho + \begin{bmatrix} \cos(\theta + \psi) & \sin(\theta + \psi) \end{bmatrix} tr). \quad (4.86)$$

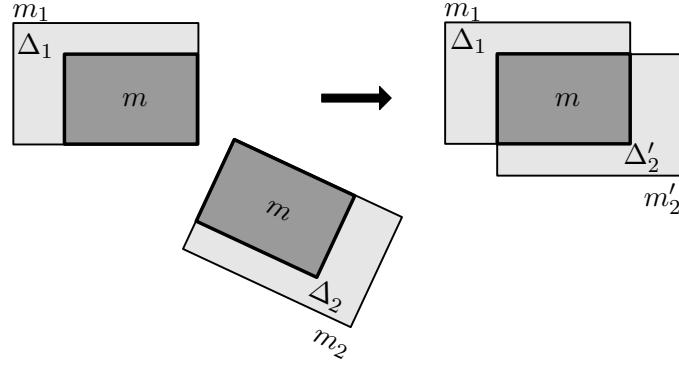


Figure 4.43: For two given maps, m_1 and m_2 , m is the overlap of the maps. Δ_1 and Δ_2 are non overlapping parts of each map.

The main tasks which are addressed in the following sections are:

- how overlaps can be identified and
- how the transformation is calculated from equation (4.86).

4.5.3.1 Extracting Relative Orientation of Maps

To calculate the transformation, it is required to find overlaps of maps. To find overlaps between maps, they should be compared with each other. One way to compare maps is to extract features from maps and then match features against each other. In general extracting map features is not easy, however, in structured environments, where maps are composed of line segments with different sizes, features can be extracted using the Hough transform. As explained, if the Hough transform is applied to an occupancy grid map, line segments appear as local peaks and the relation of local peaks in the Hough space, corresponds to the geometry of the real world. In fact, peak points are treated as features extracted from maps.

Let us assume that \mathcal{H}_1 and \mathcal{H}_2 are the Hough images of m_1 and m_2 . According to Property 4.5.3, the internal relations of the local peaks of \mathcal{H}_1 and \mathcal{H}_2 correspond to special geometric shapes (composed of line segments) in the maps. By comparing these peaks overlaps are identified in both maps (see Fig. 4.41 as an example where

there is a full overlap). These overlaps are used to find the rotation. The only limitation with this approach is that if there are similar patterns in both maps, the extracted overlaps may not be completely accurate.

In order to find a correct solution, the key is to correctly associate the relative peaks from the two Hough transforms. This is achieved using the values of the peaks, which can be mapped to the lengths of the line segments in the original maps.

Algorithm 4.5.1 shows the peak association and rotation calculation. Inputs to the algorithm are the Hough images of m_1 and m_2 , denoted by \mathcal{H}_{m_k} where $k = 1, 2$. k is the number of robots, but here the algorithm is explained for two robots for simplicity.

v_{gate} is a parameter used to reject outlier peaks. If the difference of values of two peaks is more than v_{gate} then they are not considered as associated points.

The local peaks of Hough images of m_1 and m_2 are denoted by P_k where $k = 1, 2$ (lines 1-2). P_k is defined as

$$P_k = \{p_k^i\}_{i=1}^{n_k}, p_k^i = (\theta_k^i, \rho_k^i, v_k^i). \quad (4.87)$$

P_k is a set of n_k points, each with triplets $(\theta_k^i, \rho_k^i, v_k^i)$. The first two arguments are the Hough coordinates and the last is the value of the peak at that location in the Hough image. Set C holds pointers to associated peak points from P_1 and P_2 , defined as:

$$C = \{(i_n, j_n)\}, i = 1, \dots, n_1, j = 1, \dots, n_2, n = 1, \dots, N \quad (4.88)$$

where (i_n, j_n) means $p_1^{i_n}$ is associated with $p_2^{j_n}$. N is the total number of associated points, denoted by subscript n .

Initially C is empty (line 3). For each peak from P_1 , its candidate associated point from P_2 is found by a simple search algorithm (line 4-5). If the difference of peak values is less than a threshold (line 6-7), then points are considered to be associated

Algorithm 4.5.1 Relative Rotation.

Input: \mathcal{H}_{m_1} , \mathcal{H}_{m_2} , v_{gate}

Output: ψ

```

1:  $P_1 \leftarrow$  local peaks of  $\mathcal{H}_{m_1}$ 
2:  $P_2 \leftarrow$  local peaks of  $\mathcal{H}_{m_2}$ 
3:  $C \leftarrow \emptyset$ 
4: for  $i = 1 \rightarrow |P_1|$  do
5:    $j \leftarrow \operatorname{argmin}_j |v_1^i - v_2^j|$ 
6:    $\delta_v \leftarrow |v_1^i - v_2^j|$ 
7:   if  $\delta_v < v_{gate}$  then
8:      $C \leftarrow C + (i, j)$ 
9:   end if
10: end for
11:  $\hat{\psi} \leftarrow \frac{1}{|C|} \sum_{n=1}^{|C|} (\tan \theta_1^{i_n} - \tan \theta_2^{j_n})$ 
12:  $h_1(\theta) \leftarrow f(\mathcal{H}_{m_1}(\theta, \rho))$ 
13:  $h_2(\theta_o + \psi) \leftarrow f(\mathcal{H}_{m_2}(\theta, \rho))$ 
14:  $\Psi_o = \{\psi_o^i\}_{i=1}^l \leftarrow h_1(\theta) \circledast h_2(\theta + \psi)$ 
15:  $\psi_o \leftarrow \operatorname{argmin}_i (|\psi_o^i - \hat{\psi}|)$ 
16:  $\alpha_1 \leftarrow \psi_o - \lambda$ 
17:  $\alpha_2 \leftarrow \psi_o + \lambda$ 
18:  $\delta_\psi \leftarrow \operatorname{argmax}_\theta \max(\mathcal{H}_{m_1}^{\theta=\alpha_1:\alpha_2}) - \operatorname{argmax}_\theta \max(\mathcal{H}_{m'_2}^{\theta=\alpha_1:\alpha_2})$ 
19:  $\psi \leftarrow \psi_o + \delta_\psi$ 

```

and added to C (line 8). After finding all correspondences, the slopes of the lines are calculated based on Property 4.5.3. The average difference of slopes provides an estimate for the relative rotation (line 11). Based on experiments this estimate is not very accurate. However, the estimated rotation is used to find a better solution for the rotation.

Let us assume there exists some function $f(\cdot)$, taking an array as its input and outputting a scalar, that is applied to a Hough image at a fixed angle, θ_o . Therefore the input of $f(\cdot)$ is dependant only on the second input of the Hough image.

$$h(\theta_o) = f \left(\mathcal{H}_m(\theta, \rho) \Big|_{\theta=\theta_o} \right) \quad (4.89)$$

where $f\left(\mathcal{H}_m(\theta, \rho) \mid \theta=\theta_o\right)$ means that the input of $f(\cdot)$ is one column of the image \mathcal{H}_m which corresponds to the angle θ_o . By applying $f(\cdot)$ to the Hough images of m_1 and m_2 , we will have:

$$\begin{aligned} h_1(\theta_o) &= f\left(\mathcal{H}_{m_1}(\theta, \rho) \mid \theta=\theta_o\right) \\ h_2(\theta_o + \psi) &= f\left(\mathcal{H}_{m_2}(\theta, \rho) \mid \theta=\theta_o+\psi\right). \end{aligned} \quad (4.90)$$

where $h_1(\cdot)$ and $h_2(\cdot)$ are showing the result of applying $f(\cdot)$ on the Hough image at each angle.

$f(\cdot)$ should be designed in such a way that $h_1(\cdot)$ and $h_2(\cdot)$ have highly similar patterns. This is feasible because according to Property 4.5.1 the second argument of the Hough image after transformation is shifted by a specified amount. Therefore any Order Invariant Function (OIF)¹ like *max*, *averaging over non zero elements* or *entropy* would be a good candidate for $f(\cdot)$.

By extending (4.90) to all angles, $\theta_o = [-\pi/2, \pi/2]$, two function from images are extracted; $h_1(\theta)$ and $h_2(\theta + \psi)$ (lines 12-13). To extract ψ , which is the relative rotation between m_1 and m_2 , a circular cross correlation is used. The cross correlation must be circular because the Hough image is periodic and boundary effects in periodic signal should be avoided:

$$\Psi_o = h_1(\theta) \circledast h_2(\theta + \psi). \quad (4.91)$$

where \circledast calculates the circular cross correlation of $h_1(\cdot)$ and $h_2(\cdot)$ and outputs local peaks as candidate rotations. The subscript o denotes that the outcomes of (4.91) are initial estimates and will be refined later. The location of the peak in the correlation shows an estimate for the relative rotation between the maps.

Intuitively, if *max* is used as the function $f(\cdot)$, it will be equivalent to extracting peak

¹An OIF is a function that reordering its input arguments does not effect the output. For example $f(a_1, a_2, a_3, a_4) = (a_1 + a_2)^2 + (a_3 + a_4)^2$ is not an OIF but $f(a_1, a_2, a_3, a_4) = (a_1 + a_2 + a_3 + a_4)^2$ is.

points (which represent long walls) at each angle from both maps and finding the correlation between them.

In circular cross correlation, due to incomplete overlap between maps, process noise and periodicity, other local maxima should be taken into account as potential solutions. Therefore multiple peaks are considered and only one is accepted by a rotation verification process.

Let us assume that there are l rotation candidates defined as (line 14):

$$\Psi_o = \{\psi_o^i\}_{i=1}^l \quad (4.92)$$

In line 11, only one estimate was calculated, called $\hat{\psi}$. $\hat{\psi}$ is used in a rotation verification process. This estimate is approximate but since members of Ψ_o are highly sparse, it can help to select the best rotation from the set Ψ_o by the following relation (line 15):

$$\psi_o = \operatorname{argmin}_i (|\psi_o^i - \hat{\psi}|), \quad (4.93)$$

which means the best rotation candidate is the one closest to the only approximate rotation.

The rotation can be tuned by performing a comparison between the peak points of the Hough images of m_1 and m'_2 , the rotated m_2 by ψ_o . This means that given an initial start angle, ψ_o , the peak comparison process can be performed around the rotation angle, ψ_o and within the interval (α_1, α_2) where $\alpha_1 = \psi_o - \lambda$ and $\alpha_2 = \psi_o + \lambda$ (line 16-19).

$$\begin{aligned} \delta_\psi &= \operatorname{argmax}_\theta \max(\mathcal{H}_{\theta=\alpha_1:\alpha_2}(m_1)) - \operatorname{argmax}_\theta \max(\mathcal{H}_{\theta=\alpha_1:\alpha_2}(m'_2)), \\ \psi &= \psi_o + \delta_\psi \end{aligned} \quad (4.94)$$

where $\mathcal{H}_{\theta=\alpha_1:\alpha_2}(map)$ is the Hough image of the input map over the specified domain

of angles (θ). This means the Hough image is prepared around ψ_o with the interval of (α_1, α_2) , where λ is an arbitrary small interval in which the resolution of the Hough image is higher than 1° resolution used in the previous step. max function operates the same way as $f(\cdot)$ does in equation (4.89). δ_ψ is the amount of the tuning angle added to ψ_o and ψ is the tuned rotation.

4.5.3.2 Extracting Relative Translation of Maps

Now assume that \mathcal{H}_1 and \mathcal{H}'_2 are the Hough images of m_1 , and rotated m_2 , m'_2 . As for Algorithm 4.5.1, local peaks of \mathcal{H}_1 and \mathcal{H}'_2 represent special geometric shapes. By comparing these peaks it is possible to find the overlaps between maps. To account for the geometric relations between the local peaks, the proposed algorithm can be done iteratively like point cloud matching. However, experiments show that if outliers are rejected by properly tuned parameters, one iteration suffices. Another advantage of matching peaks is that the lines and geometric shapes in the maps are represented as intensity points in the Hough images and generally dealing with points is much easier than with shapes.

To establish the correct peak association two types of information from the Hough images are used: the values of the peaks and the distance between peaks with the same value and orientation.

Algorithm 4.5.2 shows the peak association and translation calculation. The inputs to the algorithm are the local peaks of the Hough image of m_1 , P_1 , and the local peaks of the Hough image of the rotated m_2 , P'_2 . These two sets have the same structure as that introduced in equation (4.87), except that the prime superscript on P'_2 shows that the peaks are extracted from the Hough image of m_2 after applying the rotation. d_{gate} and v_{gate} are used to reject outliers.

Similar to Algorithm 4.5.1, as introduced in equation (4.88), set C is defined to hold associated points from P_1 and P'_2 .

Initially C is empty (line 1). α spans from -90° to 90° (line 2). At each α , there are limited (or no) local peaks. For each local peak residing at α , associated points from P'_2 are found considering the minimum difference in peak values, $|v_1^i - v_2^j|$ and distance $|\rho_1^i - \rho_2^j|$ over the range of $\mathcal{U} = \{(i, j) \subset \mathbb{N}^2 | i = 1, \dots, n_1, j = 1, \dots, n_2\}$ (line 3). While peaks of map_1 are at α , the algorithm looks for its associated peaks at δ_α , where δ_α is an interval defined as $[\alpha - \delta, \alpha + \delta]$. The reason to include this interval is to take into account possible inaccuracy of the rotation. For example, for a given peak from P_1 at angle 50° , its associated peak from P'_2 might reside at 51° because of inaccurate rotation. δ in this research is considered to be 1° .

The gate values, d_{gate} and v_{gate} , are used to reject outliers. This means that if the difference of peak values of two cells is less than v_{gate} and their distance is less than d_{gate} , then the cells are associated. If there exists corresponding peaks, they are added to C (lines 4-7).

According to Property 4.5.1, for each pair of associated points the following relation is established ($\psi = 0$ for \mathcal{H}_1 and \mathcal{H}'_2):

$$\rho'_i = \rho_i + \begin{bmatrix} \cos \theta_i & \sin \theta_i \end{bmatrix} tr_o \quad (4.95)$$

Considering this relation for all N corresponding points, the following relation can be formulated:

$$\begin{bmatrix} \rho'_1 - \rho_1 \\ \vdots \\ \rho'_N - \rho_N \end{bmatrix} = \begin{bmatrix} \cos \theta_1 & \sin \theta_1 \\ \vdots & \vdots \\ \cos \theta_N & \sin \theta_N \end{bmatrix} tr_o. \quad (4.96)$$

This equation can be written in the form of $b = Atr_o$ and the least squares error solution for this equation is (lines 10-11)

$$tr_o = (A^T A)^{-1} A^T b. \quad (4.97)$$

Equation (4.96) is an overdetermined system of linear equations. An approximate solution for this equation is the least squares solutions, presented in equation (4.97), which minimizes the L_2 -norm of the equation. In the literature, there are other methods to solve equation (4.96). For instance, Chebyshev solution [141] is an approximate solution, which minimizes the L_∞ -norm of equation (4.96). Least squares solution has the advantage that it is easy to calculate.

Algorithm 4.5.2 Relative Translation by matching peaks of the Hough images

Input: $P_1, P'_2, d_{gate}, v_{gate}, \Delta_x, \Delta_y$

Output: T

```

1:  $C \leftarrow \emptyset$ 
2: for  $\alpha = -90^\circ \rightarrow \alpha = 90^\circ$  do
3:    $(i, j) = \underset{\mathcal{U}}{\operatorname{argmin}} \left( |v_1^i - v_2^j|, |\rho_1^i - \rho_2^j| \right) \mid \begin{array}{l} \theta_1^i = \alpha \\ \theta_2^j = \delta_\alpha \end{array}$ 
4:    $\delta_v \leftarrow |v_1^i - v_2^j|$ 
5:    $\delta_\rho \leftarrow |\rho_1^i - \rho_2^j|$ 
6:   if  $\delta_v < v_{gate}$  and  $\delta_\rho < d_{gate}$  then
7:      $C \leftarrow C + (i, j)$ 
8:   end if
9: end for
10: Calculate  $A$  and  $b$  from  $C$  based on (4.96).
11:  $tr_o \leftarrow (A^T A)^{-1} A^T b$ 
12:  $T \leftarrow \underset{\mathcal{S}}{\operatorname{argmin}} \{H(J(m_1, T_{x, y}(m'_2))\}$ 

```

The calculated translation can be tuned to provide more accurate results. This is done using image entropy. Image entropy is defined as:

$$H(\text{map}) = - \sum p \log_2(p), \quad (4.98)$$

where p is the normalized histogram of map . Entropy is a statistical measure of the randomness associated with an image and is usually applied to characterization of image texture. A translation is desired which minimizes the entropy of aligned maps. To do this optimization, an exhaustive search in the neighbourhood of the approximate translation is done. It is challenging to use guided search algorithms

like genetic algorithms because in many cases the best solution may occur very close to very poor solutions. The size of the search space is determined by experience based on the performance of the approximate translation method. In addition, if the total allowable time for searching can be determined, the size of the search space can be specified such that the search is completed within the allotted time. Usually a search space with a radius of ten cells around the current approximate solution is acceptable considering the required time constraints and accuracy of the previous result.

The following relation gives the best translation vector (line 12):

$$tr = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \underset{\mathcal{S}}{\operatorname{argmin}} \left\{ H \left(J(m_1, T_{x,y}(m'_2)) \right) \right\} \quad (4.99)$$

$$\mathcal{S} = \{(x, y) \in \mathbb{N}^2 | \delta_{x_o} - \Delta_x < x < \delta_{x_o} + \Delta_x, \delta_{y_o} - \Delta_y < y < \delta_{y_o} + \Delta_y\} \quad (4.100)$$

where m'_2 is the rotated m_2 by ψ and $T_{x,y}(m)$ means that the input map is translated according to the values of x and y . $map_{12} = J(map_1, map_2)$ means both input maps, map_1 and map_2 are fused using equation (4.67) and are in the same coordinate frame according to the coordinates of the first input, map_1 . \mathcal{S} is the search space, defined as a rectangle centered at $(\delta_{x_o}, \delta_{y_o})$ with dimensions $2\Delta_x \times 2\Delta_y$. tr is the tuned translation vector.

4.5.3.3 Verification of Results

After finding the transformation matrix, a verification is performed. The verification, which was introduced in Section 4.2, measures the similarity of two maps, m_1 and m'_2 , where m'_2 is the transformed m_2 .

4.5.4 Advantages and Disadvantages

The proposed solution takes robots' maps to the Hough space and performs map matching. Using important properties of the Hough space, it is shown that map merging can be done faster and more robustly. Experimental results support this claim.

It should be emphasized that the accuracy of any multiple-robot SLAM solution which is based on the map merging, depends on the accuracy of the individual local maps of the robots. Therefore to have a better and consistent global map, it is important to use a robust SLAM solution to generate local maps. In this research individual robots are performing SLAM using an improved grid mapping [43].

4.5.5 Contribution

This work presents a solution to this problem using the Hough transform that will be shown to be fast and accurate. Individual robots perform robust and accurate view-based SLAM using dense laser ranger measurements [43] to generate a consistent local map of the environment. At the multi-robot level, features from local maps are extracted using the Hough transform. These features are used to calculate the relative transformation of maps and fuse them to generate a global map. It will be shown that it is more efficient to perform the map merging operation in the Hough space because of the unique representation of the obstacles in an occupancy grid map. As a result, there is no need to perform any extensive searching and the map merging algorithms scales linearly with map size.

4.5.6 Experiment

In this section, first an experiment on a dataset is explained in detail. Then an experiment is performed in Gazebo with four robots on a simulated environment.

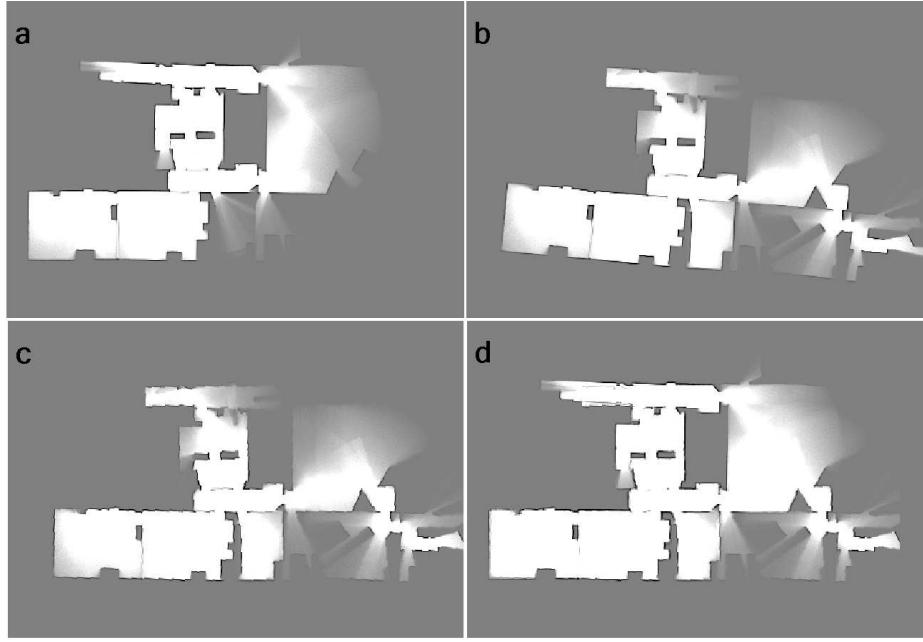


Figure 4.44: **a)** map_1 , **b)** map_2 , **c)** map_2 after tuned rotation, **d)** two maps after transformation.

Then an experiment with two robots in a real-world is presented and at the end another experiment with three robots is demonstrated.

4.5.6.1 Case1: Radish Dataset

The first experiment is performed on Radish Fort AP Hill data set [136]. This is an open source and well-known data set which is used as a benchmark data set for view-based multiple-robot SLAM. The raw laser ranger and encoder data are recorded in standard Player [137] format and fused by particle filtering [28].

Fig. 4.44-a and Fig. 4.44-b show the two maps before fusion. The maps have about 5° rotation. The proposed method is used to find the relative rotation between the maps.

By applying the *Hough Rotation* block of Algorithm 4.5.1, an approximate rotation is extracted as $\hat{\psi} = -15^\circ$.

Then the Algorithm 4.5.1 continues with *Multiple Hypothesis*. Fig. 4.45-a shows $h_1(\theta)$ and $h_2(\theta + \psi)$ in blue and red colours. $h_1(\theta)$ is the output of $f(\cdot)$ on m_1 and $h_2(\theta + \psi)$

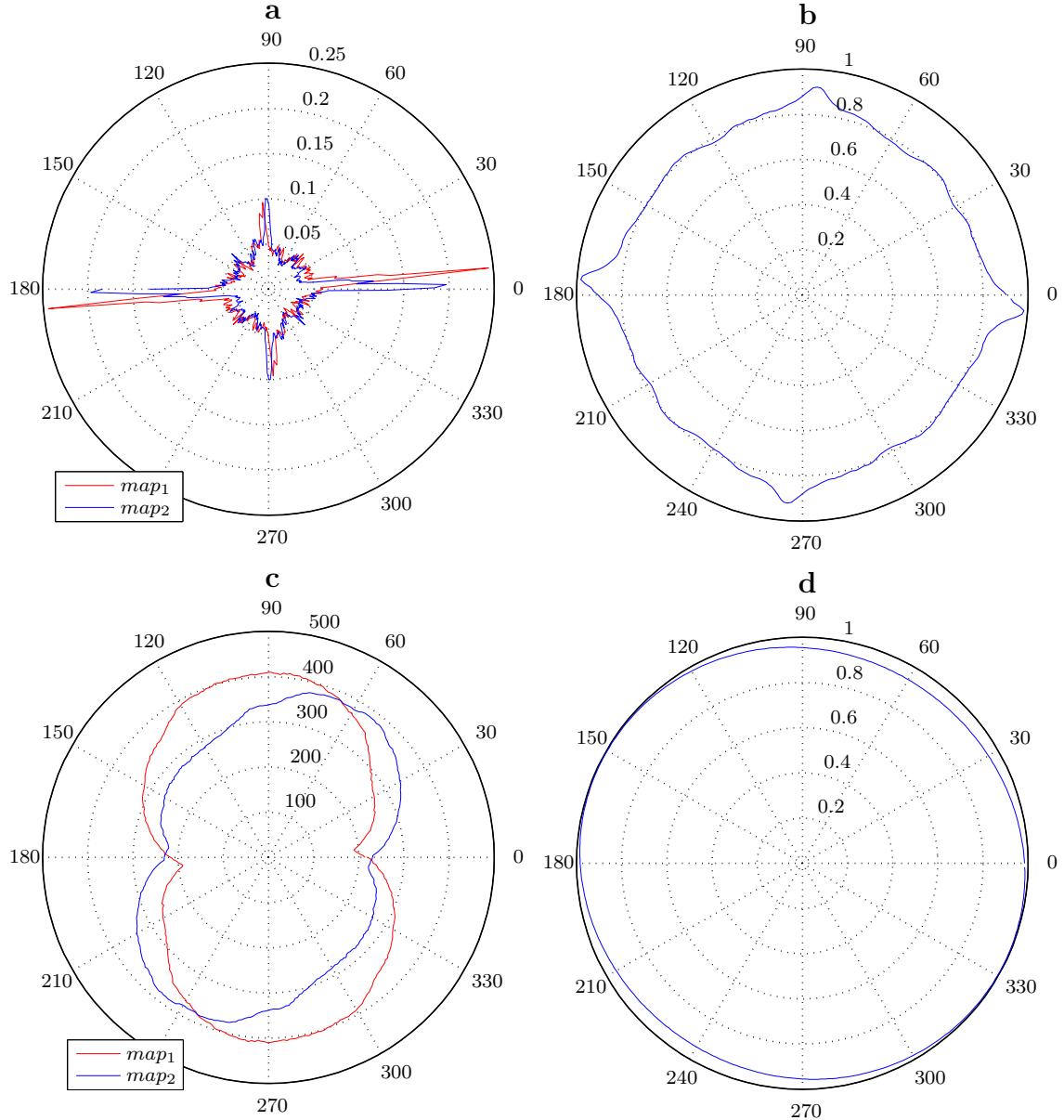


Figure 4.45: **a)** $h_1(\theta)$ and $h_2(\theta + \psi)$ when $f(\cdot)$ is $\max(\cdot)$, **b)** Result of circular cross correlation. Four local peaks are at -5° , 85° , 175° and -95° , **c)** $h_1(\theta)$ and $h_2(\theta + \psi)$ when $f(\cdot)$ is an average function, **d)** Result of circular cross correlation. Two local peaks are -30° and 150° .

is the output of $f(\cdot)$ on m_2 , where $f(\cdot)$ is $\max(\cdot)$ function. To find ψ_o the result of the circular cross correlation has been depicted in Fig. 4.45-b. The result has four local peaks, $\Psi_o = \{-5^\circ, 85^\circ, 175^\circ, -95^\circ\}$. These peaks are considered as potential candidates for ψ_o . The approximate rotation which is $\hat{\psi} = -15^\circ$ is close to -5° .

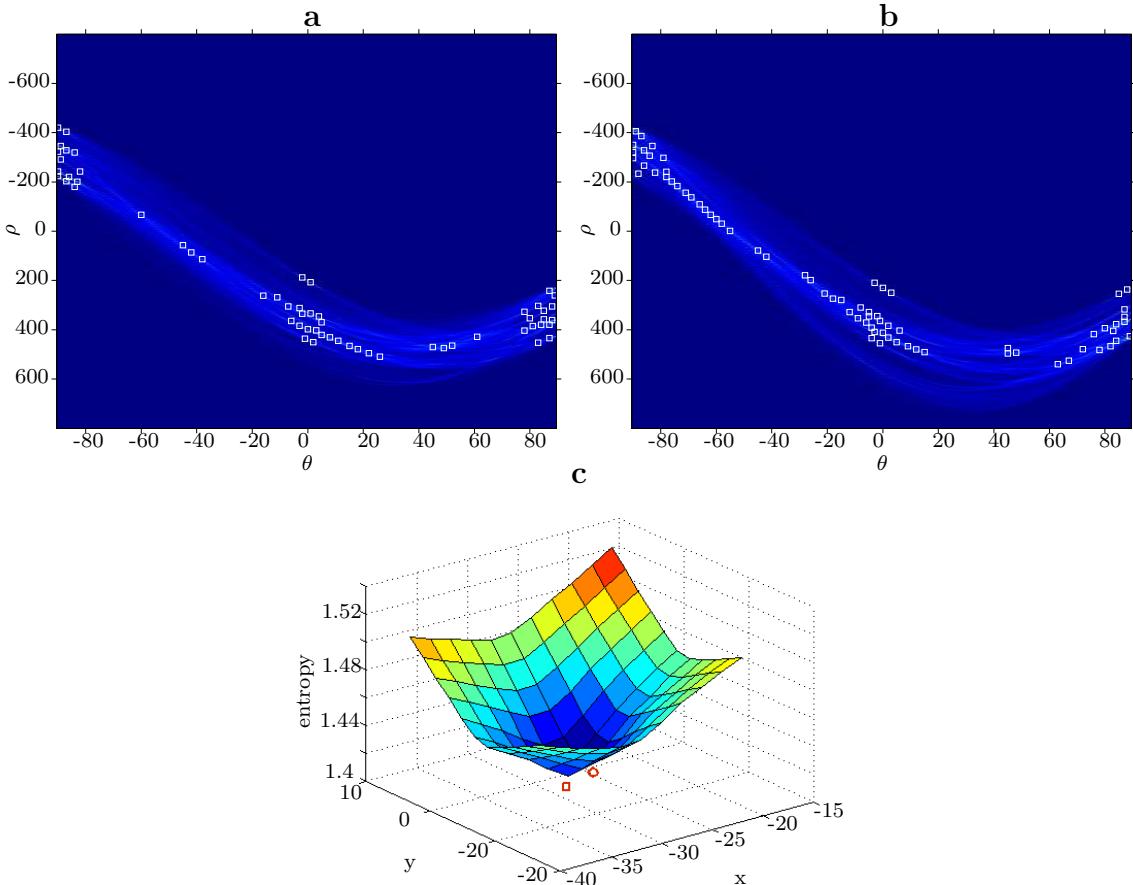


Figure 4.46: **a)** Hough image of map_1 with peak points marked with white squares, **b)** Hough image of rotated map_2 with peak points marked with white squares, **c)** Tuning translation by image entropy. The point shown by a circle is the initial estimate of the translation. The point shown by a square is the tuned translation.

Based on equation (4.93) the rest of the experiment continues with $\psi_o = -5^\circ$.

To compare the effect of other alternatives for $f(\cdot)$, *averaging over non zero elements* of the given input vectors has been examined. Fig. 4.45-c and Fig. 4.45-d show $h_1(\theta)$ and $h_2(\theta + \psi)$ and the result of correlation. This time the result has bigger offset from the required estimated value. *Entropy* of the input vector has been also evaluated which has an offset of the order of averaging function. $\max(\cdot)$ produces better results and is used for other experiments in this work.

After the tuning method ψ becomes -5.5° . The result of the final alignment is shown

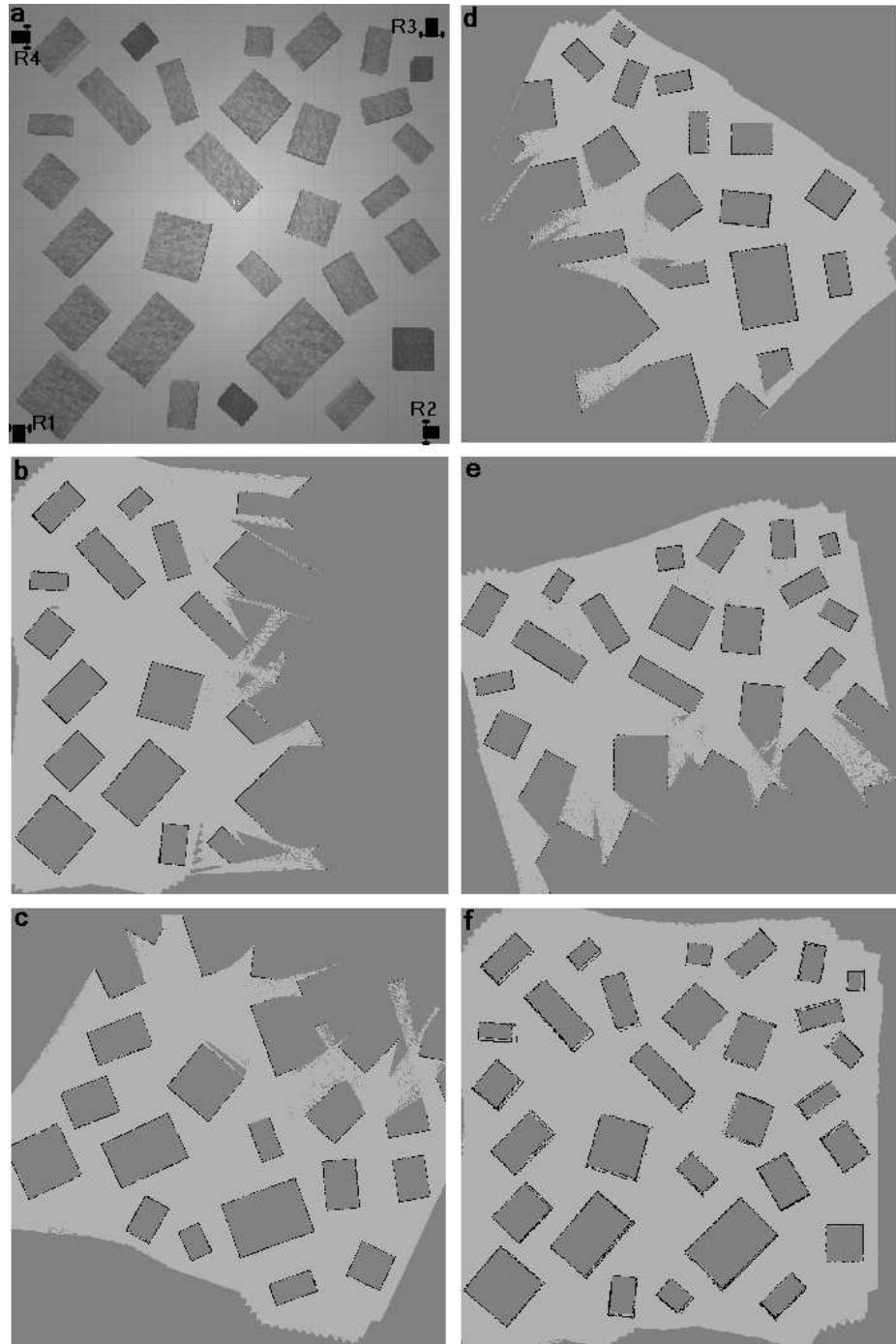


Figure 4.47: **a)** Simulated Gazebo world. Four Erratic robots map the world producing four partial maps, **b)** map by robot *R1*, **c)** map by robot *R2*, **d)** map of robot *R3*, **e)** map of robot *R4*, **e)** fused four maps to the coordinates of *R1*.

in Fig. 4.44-c.

Now the next step which is finding the translation is started. In this step first Hough images of m_1 and rotated m_2 , m'_2 are generated. These two images are shown in Fig. 4.46-a and 4.46-b respectively. Using the data association of Algorithm 4.5.2 and solving equation (4.97), the initial estimate for the translation tr_o is $[23 \ 1]^T$. (v_{gate} and d_{gate} for all experiments are 10 and 30, set by experiments.) Then using the exhaustive search and evaluating the image entropy the translation is tuned to be $[27 \ 2]^T$. Fig. 4.46-c shows the convergence of the image entropy. The initial estimate is shown by a circle and the tuned estimate is shown by a square. Finally Fig. 4.44-d shows the final alignment of both maps. The verification index, defined in equation (4.27) is 94%.

4.5.6.2 Case2: Simulation in Gazebo with Four Robots

This experiment is performed in ROS (Robot Operating System)-Gazebo [142] simulation environment. The simulated world is shown in Fig. 7.8-a, composed of 30 blocks with different sizes and orientations. Four Erratic robots start to explore and map the world using the gmapping Algorithm [43].

As figures 7.8-b to 7.8-f show, each robot covers almost half of the world. The amount of overlap between the maps is shown in Table 4.6. (numbers are approximate.). Fig. 7.8-e shows four final maps fused using the proposed algorithm.

Table 4.6: Percentage of approximate overlaps between maps.

Robot	R_1	R_2	R_3	R_4
R_1	-	61	5	48
R_2	61	-	52	15
R_3	5	52	-	42
R_4	48	15	42	-

4.5.6.3 Case3: Two CoroBot Robots

A real world experiment is performed with two CoroBots in an indoor environment in the basement of Head Hall, UNB. The total area is 81.02 m^2 and the lengths of robot trajectories are 22.5 m and 25.9 m.

Fig. 4.48-a and Fig. 4.48-b show the maps of the robots. Although the area of this experiment is not very large, it is important for two reasons. First as the maps show, the required rotation is close to 90° . Second the overlap between two maps is approximately 25.4 m^2 which is 31% of the total area and this is important to show the effectiveness of the algorithm with small overlaps (The method proposed in [31] fails in this experiment due to small overlap). The overlaps in two maps are shown by ellipses. In both maps, there are non-overlapping corridors with almost the same size (15.5 m and 12.2 m). But the algorithm is capable of rejecting them as outliers and finding the transformation based on the overlap. Fig. 4.48-d shows final maps after fusion.

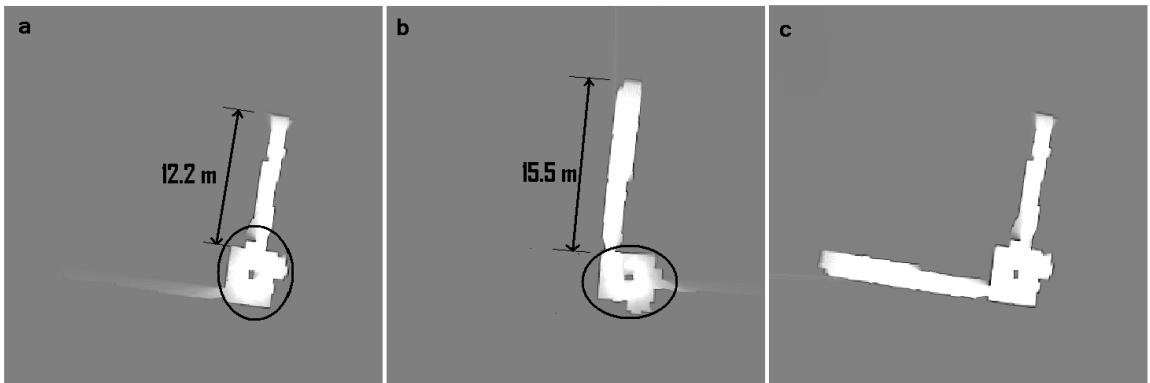


Figure 4.48: The experiment with two robots. **a)** map_1 , **b)** map_2 , overlaps of two maps are shown by ellipses, **c)** fused maps.

4.5.6.4 Case4: Three CoroBot Robots

This experiment is performed in a larger environment, with a coverage area of approximately 600m^2 and three agents are involved. Trajectories of robots are approximately

$60m$, $35m$, and $55m$. By merging maps, loop closure happens successfully.

Figures 4.49-a, b, and c show three partial maps. Maps of Figures 4.49-b and c are fused to Fig. 4.49-a. Overlaps of Fig. 4.49-a with other two maps are enclosed inside polygons. The fused global map of the environment is shown in Fig. 4.49-d.

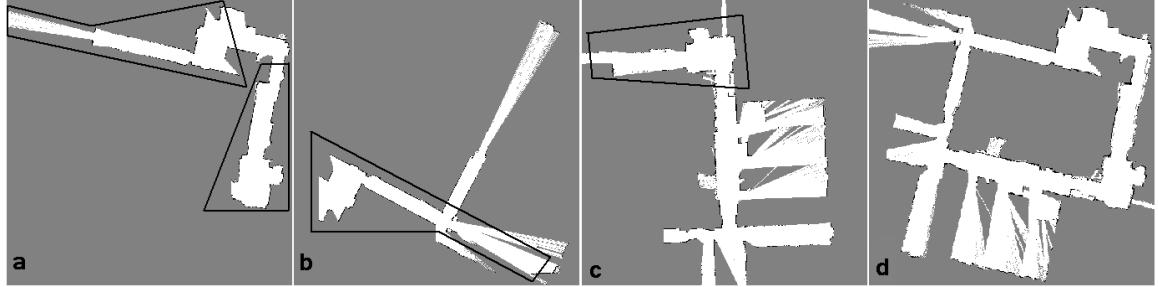


Figure 4.49: Three partial maps are fused together to generate a global map. (a) is the base map where (b) and (c) are fused to that. Overlaps are marked with polygons. (d) Final fused map which depicts loop closure.

4.5.6.5 Discussion: Tuning Parameters

Throughout the algorithm, a few parameters are introduced which need to be tuned for good performance. Table 4.7 lists these parameters. In this work, these parameters have been tuned by trial and error; however, it is possible to use non-derivative optimization methods which is out of the scope of the work. Repeated experiments show that changing these nominal values for about $\pm 15\%$ does not affect the quality of the output significantly.

Table 4.7: List of parameters with their nominal values.

Block	Parameter	Value
Hough Rotation	v_{gate} : peak value threshold	10
	λ : rotation tuning interval	1°
Hough Translation	v_{gate} : peak value threshold	10
	d_{gate} : distance value threshold	30
	Δ_x : translation tuning interval	10
	Δ_y : translation tuning interval	10
Similarity Index	verification threshold	92 %

4.5.6.6 Discussion: Comparisons

As mentioned, a major benefit of the Hough peak matching for map fusion is the low processing time requirement and its robustness. The proposed method is compared with ARW map merging [2], Hough spectrum [31], map segmentation [121] and Fourier-Hough registration [143] methods.

Table 4.8: Normalized processing time and efficiency comparison of four experiments with three other methods, ① Radish data set, ② Gazebo world with four robots ③ two CoroBots, ④ three CoroBots.

Experiment	①	②	③	④	①	②	③	④
Method	Processing (s)				Verification (%)			
Hough peak matching	14	12	11	13	94	92	92	94
Map segmentation [121]	105	68	83	106	95	91	92	94
ARW map merging [2]	168	140	150	152	93	90	88	92
Hough spectrum [31]	16	16	<u>fails</u>	17	92	89	<u>fails</u>	90
Fourier-Hough transform	19	17	<u>fails</u>	16	88	89	<u>fails</u>	87

ARW map merging and map segmentation were explained in Section 4.2. The Hough spectrum approach is based on correlation of Hough images and there is no assessment on overlaps between maps. In this method, a spectrum function, originally proposed in [144] by Censi *et al.*, is defined on the Hough image. The function is applied to both of Hough images of maps, then the correlation of the results is used to find potential rotations. There are multiple candidates which all are processed to find the translation. For each candidate a translation is calculated and at the end, only one rotation and translation are accepted. Carrying over all rotation candidates requires more processing time and computational power and makes the algorithm slow. For each rotation, the translation is found using correlation of projections of the Hough images along X and Y directions. For example, if projections of the Hough image of map m_1 are shown by p_{x_1} and p_{y_1} and for the rotated map m_2 by p_{x_2} and p_{y_2} , then the peak of correlation of p_{x_1} and p_{x_2} results in the translation along x and the peak of correlation of p_{y_1} and p_{y_2} generates the translation along y . This approach is

effective if projections have enough similar patterns, and this similarity happens only when there is a significant amount of overlap between maps. Therefore, finding the translation fails in maps with less overlaps.

In the Fourier-Hough registration method [143, 145, 146], first a 1D phase correlation is applied to the Hough images of two maps to find the relative orientation. Once the maps are aligned using the calculated orientation angle, a 2D convolution using the Fourier transform is applied to the aligned maps to calculate the translation.

Table 4.8 summarizes the comparison of the processing time and verification for all three experiments (results are normalized for one pair of maps). For the third experiment, Hough spectrum approach fails due to having less overlaps. For the same reason, the Fourier-Hough fails to calculate the transformation. As the results show, the proposed method operates fast and the verification index shows the accuracy of the results.

Experiments with different overlaps show the effectiveness of the proposed algorithm. In the first experiment, Radish data set, there was an approximate overlap of 70%. In the Gazebo simulation, the overlap between robots is shown in Table 4.6. In the last two real world experiment, the overlap were approximately 31% and 38%.

4.5.7 Summary

Multiple robot map merging in the Hough space has been presented that is fast and robust. The proposed algorithm is designed to work in structured environments. The peaks of the Hough image are used to model the real world and find overlaps between maps. Then the overlaps are used to calculate the relative transformation. Other properties of the Hough transform are used to boost the speed and efficiency of the proposed method. Experiments on data sets and collected data show the effectiveness of the method. Efficiency and processing time comparison with other established methods show that the proposed method is both fast and accurate.

In future work, an efficient iterative peak matching will be investigated to solve for all variables of the transformation matrix at once to further increase the speed. Also choosing the appropriate time or criteria to fuse maps optimally is another open problem to be investigated.

4.6 Summary of Map Merging

In this section, the summary of the work on map merging is presented. Identified problems in multiple-robot SLAM are summarized as follows:

- **Finding relative transformation matrix:** In multiple-robot SLAM, each robot tries to integrate all of the local maps provided by individual robots to generate a global map of the environment. However, the required transformation matrices which relate these maps to each other are unknown. For this problem, four solutions were proposed:
 - Map segmentation: This algorithm is performed by applying Canny edge detection and introducing a smoothing method to extract unique characteristics of occupancy grid maps. These unique characteristics are used to find the relative transformation matrix.
 - Neural Networks Approach: This method is a neural network-based map clustering approach to scale down and extract features of occupancy grid maps. The extracted features are used to calculate the transformation matrix. This method is based on the competitive self-organizing maps. There is no known similar research or application.
 - Probabilistic generalized Voronoi diagram: This method is a novel probabilistic map fusion algorithm based on the GVD. In this method, morphological operations are used to build probabilistic generalized Voronoi

diagrams for occupancy grid maps. Then PGVDs are cross-matched to find the transformation matrix.

- **The Hough peak matching:** In this approach, maps are transformed into the Hough space. Using the properties of the transformation elements in the Hough space and by matching local peaks of the Hough images, the required alignments are calculated robustly.
- **Updating maps:** Once the relative transformation is found, a procedure is required to fuse local maps. The resulting map should integrate all information from given local maps.

For this problem, an entropy filter is proposed. After finding the transformation matrix, an entropy filter is used to update occupancy grid maps.

- **Propagating the uncertainty of the relative transformation:** There is an uncertainty associated with the relative transformation matrix represented in the form of a covariance matrix. Updating the maps should be performed using the covariance matrix. For this problem, LUP is proposed. Given the transformation matrix and its uncertainty as a covariance matrix, a formulation is proposed to consider the effect of the uncertain transformation on the transformed map.

- **Computational demand:** Robotics applications are usually realtime. Thus, it is very important to design an algorithm capable of solving the above-mentioned problems with minimum time and memory requirements. Finding the transformation matrix is the most time-consuming part. As the comparisons presented in each chapter show, PGVD and Hough peak matching are the fastest methods. The neural networks approach has a moderate speed while map segmentation is the slowest. Extensive experimental results show that the Hough approach operates in any kind of structured environment. This is because the Hough image is a complete model of the world in an abstract form

which makes further processing easier. The PGVD approach is also highly accurate but its accuracy is dependent on the method used to generate the GVD. Comparison of the proposed solutions with others show that the former are faster and more robust.

Chapter 5

Particle Filter-based Multiple-robot SLAM

This chapter presents a particle filter-based solution for multiple-robot SLAM. This solution proposes an efficient and feasible algorithm. Different problems in multiple-robot SLAM are taken into account and a general solution is presented and then implemented using particle filtering. Background and map merging results from the previous chapters are used to address some of the problems. The rest of the chapter is organized as follows:

- Section 5.1 highlights the system overview including problems, current solutions, proposed solutions, and the advantages of our solution;
- Section 5.2 introduces a particle filter for multiple-robot SLAM;

5.1 System Overview

As mentioned in the previous chapters, multiple-robot SLAM algorithms in real-world applications have to deal with many problems and limitations on resource. In the rest of the section, some of these limitations and problems are briefly reviewed.

5.1.1 Problem Statement

Multiple-robot SLAM has benefits such as being more robust to failures and being faster in mapping unknown environments. In the previous chapter, map merging as a solution to this problem was introduced, which requires sharing information and processed data such as the maps of the robots among the robots. It is also possible to share unprocessed data such as raw laser observations and develop a global map by processing them. Compared with the map merging solutions, sharing raw data has the advantage of being more flexible when updating the incremental mapping and localization, since access to all unprocessed data is available; however, there are some problems and limitations which will be listed and explained.

5.1.1.1 Relative Poses of Robots

To start mapping and localization by a team of robots, initial poses of the robots or their relative poses must be known. Otherwise it is impossible to integrate all measurements into a global map. In general, relative poses are unknown; however, it is possible to calculate them by comparing maps of individual robots, as explained in the map merging chapter. It is also possible to calculate the relative poses from the encounters of the robots. The latter solution requires the robots to meet at some point which imposes a limitation on the paths of the robots. In this work, the results from the previous chapter will be used to calculate the relative poses of the robots.

5.1.1.2 Uncertainty of the Relative Poses

Once the relative poses are known, observations from all robots can be integrated to generate a global map. However, the transformation matrices, which represent the relative poses of the robots, are in general not deterministic. The uncertainty associated with the relative transformation matrices should be propagated onto the global map to preserve the probabilistic nature of the map.

5.1.1.3 Line-of-sight Observations

During encounters, robots can see each other through line-of-sight observations. These encounters impose constraints on the poses of the robots, which if it is incorporated to the solution, can improve the accuracy of the localization and mapping.

5.1.1.4 Closing Loops

Loop closure in the multiple-robot SLAM requires identifying previously observed locations. A location might have been mapped by a robot, but the loop closure through that location might happen by a different robot. To perform the loop closure at the team level, a global and consistent map should be maintained and updated incrementally.

5.1.1.5 Updating Maps and Poses

In a multiple-robot SLAM algorithm, it should be possible to update the map and the pose of each robot when new information is available. To do this, correlation of the poses through the direct encounters and through the global map should be taken into account. Moreover, when a map is updated by a robot, the poses of other robots should also be updated, if applicable.

5.1.1.6 Communications

Communication problems such as black outs, delays, and out-of-sequence packets are major problems which affect the performance of real-time multiple-robot SLAM algorithms.

5.1.2 Description of the Proposed Method

In the multiple-robot SLAM solution, a general solution for mapping and localization by a team of robots is presented. The proposed algorithm is implemented and verified

using particle filtering. The presented solution starts with the assumption that the relative poses are known, then the assumption is removed and a general solution using the results of the previous chapter is presented. At the end, the developed algorithm is verified in a simulated environment.

5.1.3 Summary

This section presented an overview of the proposed method for particle filter-based multiple-robot SLAM. Main problems in multiple-robot SLAM were briefly reviewed. Then the general description of the proposed solutions was presented. In the following sections, the proposed solutions are explained with more details.

5.2 Multiple-robot SLAM

This section studies the multiple-robot SLAM problem. Two different scenarios are presented, which are the multiple-robot SLAM with known relative poses of the robots and the multiple-robot SLAM with unknown relative poses. In the known relative poses case, it is assumed that the robots initially know each others' poses. Then the solution is extended for the scenario where the robots are unaware of their initial poses. For the latter scenario, results from the previous chapter are used to calculate the relative transformation between the robots. Implementations based on particle filtering are presented for both scenarios. Line-of-sight observation and communication issues are also studied for these scenarios.

5.2.1 Problem Statement

Multiple-robot SLAM is a complicated problem which requires coordination and co-operation of all robots to achieve the desired results. Key problems in multiple robot-SLAM have already been identified and reviewed. For some of the problems,

such as finding the relative transformation between the robots, the related solutions were reviewed. Briefly, multiple-robot SLAM can be stated as *finding the posterior over poses of all robots and the global map of the environment, given all measurements and control actions of the robots*. This solution should deal with unknown relative poses of the robot, communication and scalability issues, and the nonlinearity of the system.

5.2.2 Problems with Existing Methods and Motivations

There are few works formulating and performing multiple-robot SLAM with nonlinear filters. Some solutions, such as [20, 21, 107, 110, 111, 147], are based on GraphSLAM or different variations of the Kalman filter which were already reviewed in Chapter 3. In this section, problems with key existing methods using particle filtering are briefly reviewed.

In Thrun’s solution [106] for multiple-robot SLAM, which is based on particle filtering, it is assumed that the relative poses of the robots are known or they start from nearby locations. This assumption is limiting and does not hold in general.

Howard’s solution [28], which is also based on particle filtering, ignores (or engineers away) observations between robots. Moreover, he assumes that trajectories of the robots are decoupled, which in general is not a valid assumption. However, the assumption was violated a few times when the global map was used to calculate the measurement model. Use of the global map forces the robots’ trajectories to be correlated through the map. The proposed solution relies on direct encounters between robots to calculate the relative transformations, which is a limiting assumption and requires an additional layer of challenge to coordinate the robots’ motions. Also A. Howard ignores the uncertainty of relative transformations and assumes that the transformations denoting the relative poses of robots are deterministic. However, he proposes the idea of virtual robots to integrate past measurements after encounters

between robots. Using the virtual robot concept, once robots meet each other, measurements are processed in reverse temporal order to integrate past observations with the global map.

Gil *et al.* [22] proposes a feature-based multiple-robot SLAM. The focus of the work is on managing the data association of visual features, assuming the visual descriptors have been affected with noise. The proposed method has been verified in simulated environments, where robots can send their information to a central agent. Moreover, it is assumed that initial relative poses of the robots are approximately known and the direct encounters between the robots are ignored.

Carlone *et al.* [109] also apply a particle filter for multiple-robot SLAM as in [28]. Their contribution considers the uncertainty of the relative transformation in the mapping and using grid mapping [43] for particle filtering. The main disadvantage of the work is that it does not hold a joint posterior for trajectories and relative poses of the robots. In the proposed solution, robots exchange data only in encounters. This way, having access to a communication channel is not required at all times, but it can cause buffer overflow or it can occupy all processing resources for processing all buffered data. Also, in this work all encounters are taken into account to improve relative poses, unlike the work by Howard [28] which only takes into account the first encounter. In this work only one experiment with two robots is presented. This work also suffers from exponential space complexity.

Multiple-robot SLAM is a complicated problem due to the challenges inherent to multi-agent systems; however, by using other resources, available in a multiple-robot environment, such as line-of-sight observations, it is possible to handle these challenges efficiently. The work in this section is motivated by the fact that incorporating all available information into the multiple-robot SLAM algorithm will eventually result in more effective and accurate mapping and localization.

5.2.3 Notation and Definition

The notation in this chapter follows the standard presented in *Probabilistic Robotics* [3].

The pose of a robot from time 1 to time t is shown by the sequence $\{x_1, x_2, \dots, x_t\}$.

This sequence is shown in the compact form of $x_{1:t}$. Also, since multiple agents are involved, the identification number of each robot appears as a superscript in the state variable. Therefore, the following sequence shows the state of the i^{th} robot from time 1 to time t .

$$x_{1:t}^i \equiv \{x_1^i, x_2^i, \dots, x_t^i\}, \quad (5.1)$$

where $i = 1, \dots, n$ and n is the number of the robots. Respectively, the observations made by the i^{th} robot and the control signals which drive the robot at the same times are shown as

$$\begin{aligned} z_{1:t}^i &\equiv \{z_1^i, z_2^i, \dots, z_t^i\}, \\ u_{1:t}^i &\equiv \{u_1^i, u_2^i, \dots, u_t^i\}. \end{aligned} \quad (5.2)$$

Notice that sometimes the robots' identification numbers are represented by alphabetical characters. Moreover, in most equations, derivations start with the trajectory of the robots and when appropriate, the Markov assumption is applied to them.

The map of the environment at time t is often shown by m_t . Similarly, $m_{1:t}$ denotes the map at different times from time 1 to time t . Maps in this chapter are usually occupancy grid maps, unless otherwise specified. When the map has no superscript, it simply indicates that the map is a global map generated by all robots involved in the mapping. When a map has a superscript, it shows that the map is made by the robot whose identification number is given in the superscript. For example, m_t^a points to the map made by robot a at time t and m_{t-1}^b specifies the map built by robot b at time $t - 1$. m_t^{ab} shows the map built by robots a and b . If in a team there are only two robots a and b , then m_t and m_t^{ab} are equivalent.

In particle filtering, each particle holds state variables of the system. The state variable of each particle is identified by a number in parentheses. For example, $x_t^{a(i)}$ shows the pose of robot a at time t held by the i^{th} particle, $m_t^{ab(j)}$ points to the map built by robots a and b at time t maintained by the j^{th} particle. The set of all particles of the filter is shown by symbols S or P. For example, S_{t-1}^b shows set of all particles representing the state of robot b at time $t - 1$. A robot's state, map, pose, or particle set in its own local coordinates (before being transformed to a global coordinates) is described by the sign $\hat{\cdot}$ above it. For instance, $\hat{x}_t^{b(i)}$ shows the pose of robot b at time t held by the i^{th} particle, in its own local coordinates.

In this chapter, the Bayes network is used to explain the transition of the states. Fig. 5.1-a shows an example of a Bayes net for SLAM for two robots, a and b . Each variable or input has been enclosed in a circle. The circles with grey colour are inputs such as control actions or measurements. The circles with white colours are unknown state variables. Arrows show the dependency of the circles on each other. For example, x_t^a depends on x_{t-1}^a and u_t^a as shown by the arrows. In turn, z_t^a depends on x_t^a and the map of the world. On the other hand, the relation between z_t^a and the real world is important. Although the measurements are made from the physical world, in the literature, following the tradition, there is no “distinction between the physical world and the map”, as Thrun *et al.* state [3]. Therefore, the map developed so far is used to calculate the likelihood of the measurement. In other words, when a measurement is made, the current model of the world is not available. Thus, the measurement is dependent on the map from the previous time, m_{t-1}^a . Once the measurement is used to update, for example, a particle filter, then the measurement is used to make the map. For instance, z_t^a is processed using x_t^a and m_{t-1}^a and then used to make m_t , where m_t is used in the cycle of $t + 1$. These dependencies make the Bayes net very complicated. Therefore, often the time indices of maps in the Bayes nets are dropped and all maps are shown with only one circle. For example, the net

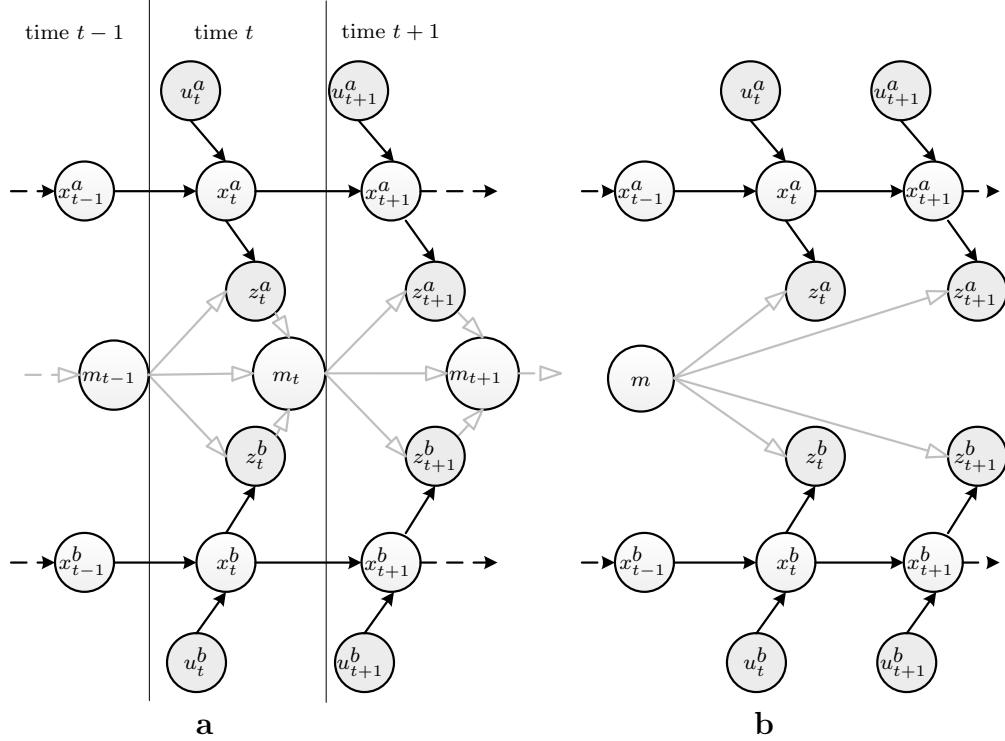


Figure 5.1: The Bayes net for SLAM. The relations between variables are shown by arrows. The relation between the map and measurements is complicated. Due to unavailability of a model from the physical world, measurements are conditioned on the map. Once measurements are processed, they are added to the map and used for processing the measurement model in the next time interval. In the literature, for the sake of brevity, the Bayes net in **a** is replaced with that in **b**.

shown in Fig. 5.1-b is used instead of that shown in Fig. 5.1-a.

Without loss of generality, it is assumed that the information from all robots is integrated into the coordinates of the first robot. For example, if there are two robots, *a* and *b*, on the team, then the map and pose of robot *b* are transformed into coordinates of robot *a*. In other words, all derivations are performed with respect to robot *a*. A simple scenario to implement such a paradigm is to send all data to a ground station. Similar to the previous chapter, the state of a robot includes 2D Cartesian coordinates and the orientation of the robot; however, all derived relations are applicable to robots with 3D translation and rotation. The observations include range and bearing measurements and later in the subsections related to the experimental

results, details of the measurement and motion models will be mentioned.

Definition 5.2.1. *Line-of-sight observation (direct encounter): when two robots can see each other, the observations are called line-of-sight observations. A partial line-of-sight observation occurs when only one robot sees the other one. A full line-of-sight observation occurs when both robots can see each other. Line-of-sight observation is equivalent to a direct encounter of the robots.*

Definition 5.2.2. *Collective observation (indirect encounter): when two robots can see the same parts of the world, the robots have made collective observations or alternatively the robots had an indirect encounter.*

Definition 5.2.3. *Cyclic update (overconfidence): cyclic update happens when robots use a measurement repeatedly. This happens when one or multiple robots use the same information more than once.*

Definition 5.2.4. *Out-of-sequence measurements: if a measurement arrives at its destination in the wrong order, for example if it is delayed, it is an out-of-sequence measurement. This happens during buffering or communications latency problems.*

5.2.4 Description of the Proposed Method

The proposed solution starts with the assumption that the robots know their relative poses in advance. An algorithm based on this assumption is developed. Then the solution is extended to include the general case in which the relative poses are not known. These two cases are described as *known relative poses* and *unknown relative poses* as it follows.

1. **Known relative poses:** In this scenario, the relative poses of the robots are known in advance. This includes scenarios where robots are dispatched to map an environment from a nearby location, so their initial relative poses are

known, though with uncertainty. For example, consider a building inspection case, where all robots enter the building through a known entrance. To develop cooperative mapping and localization, available data resources are utilized as follows:

- **Individual robots' raw data:** The communication channels are used to share raw sensor data such as unprocessed laser measurements and odometry data. On each robot, a particle filter, which includes pose of all robots, performs SLAM and generates a global map. Sharing data happens constantly if there is no interruption in the communication channels. Otherwise data is buffered to be processed later.
- **Line-of-sight observation:** When robots can see each other while moving, line-of-sight observations are used to improve the localization of the robots. Line-of-sight observations of the robots are considered as another source of information which improves the accuracy of localization and mapping. In general, robots can see and detect each other using an onboard camera or a scanning laser ranger.

2. **Unknown relative poses:** Generally robots do not have any *a priori* information about their relative poses. Such a situation occurs when robots are deployed to map an environment at widely separated locations, for example different entrances of an unknown building. In this case, the relative poses are extracted using shared maps. In this case, we utilize available resources as follows:

- **Individual robots' map:** Since relative poses are not known, maps are used to calculate them. Maps are shared by communication channels between robots. This process runs within fixed time intervals until the relative poses are calculated. Proposed solutions in Chapter 4 are used to

calculate the relative poses of the robots.

- **Individual robots' raw data:** Once the relative poses are calculated, similar to the previous case, the communication channels are used to share raw sensor data. This data is integrated by a particle filter to generate a global map.
- **Line-of-sight observation:** As in the previous case, when robots can see each other while moving, line-of-sight observations are used to improve the accuracy of localization and mapping.

Table 5.1 summarizes the resources and their utilization for the two cases discussed: known and unknown relative poses. Note that in the case of the known relative poses, maps are not shared. In general, maps are considered as processed data and sharing processed data can cause overconfidence. However, in the case of the unknown relative poses, maps are shared and used, because other than the line-of-sight observations, which may not be available, there are no any other resources to find the transformation between the poses of the robots.

Table 5.1: Utilization of information and data for different cases.

Resource \ Case	known relative poses	unknown relative poses
raw data	✓	✓
processed data (maps)	-	✓
line-of-sight	✓	✓

5.2.4.1 Known Relative Poses

In this section, it is assumed that the relative poses of robots are given. Based on this assumption, sensor readings from all robots are integrated with a particle filter.

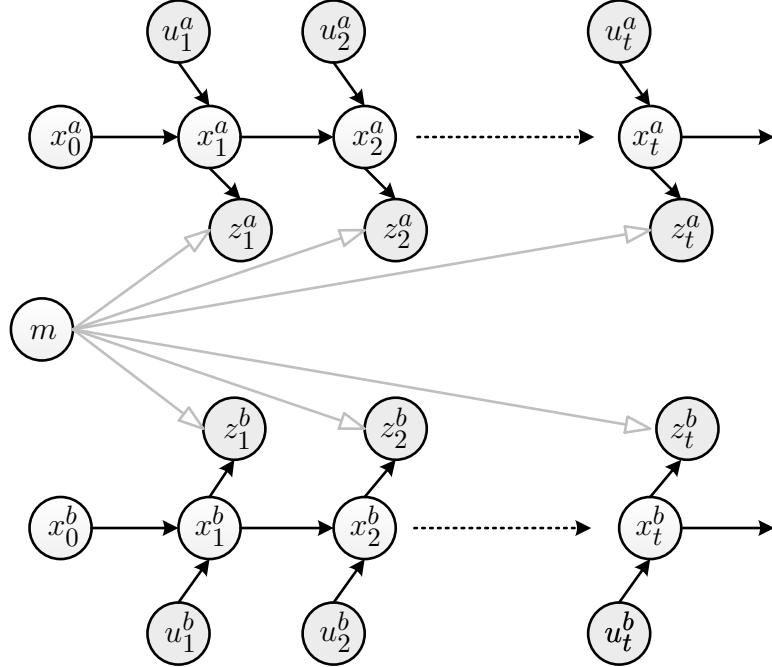


Figure 5.2: Bayes net for multiple-robot SLAM with two robots, a and b , with known relative poses and no line-of-sight observation. Black lines correspond to the state transitions of each robot. Grey lines show the relation of the map and observations.

The formulation is initially presented and derived for two robots and then extended to a team of n robots.

To begin with, in the following section, multiple-robot SLAM with known relative poses is presented with the assumption that trajectories of robots are decoupled and independent. This means that trajectories are coupled neither through line-of-sight observations nor through the global map. It is shown that in the case that robot trajectories are independent, particle weights can be updated as the product of the observations from each robot following the recursive formulation derived in Chapter 5. In the subsequent section, these assumptions are removed and the derived relations are updated accordingly.

5.2.4.1.1 Decoupled Trajectories

In this section, multiple-robot SLAM is studied assuming that the robots' trajectories are independent. In other words, line-of-sight and collective observations are ignored.

Fig. 5.2 shows the Bayes net for two robots, a and b . At time t , the poses of the two robots are shown by x_t^a and x_t^b , observations are shown by z_t^a and z_t^b and control signals are u_t^a and u_t^b . Initial values of poses are shown by x_0^a and x_0^b , respectively. m is the map of the environment which can be a feature map or an occupancy grid map. Black lines show the transition of the state of each robot. Grey lines show the relation of the map to observations.

Multiple-robot SLAM seeks to calculate the posterior over poses of all robots and the map:

$$p(x_{1:t}^a, x_{1:t}^b, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b). \quad (5.3)$$

Following the discussion in Chapter 3, the posterior in equation (5.3) is represented as follows [28]:

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ = & p(m_t | z_{1:t}^a, z_{1:t}^b, x_{1:t}^a, x_{1:t}^b, x_0^a, x_0^b) p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \end{aligned} \quad (5.4)$$

Since it is assumed that the trajectories are independent, equation (5.4) becomes

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ = & p(m_t | z_{1:t}^a, x_0^a, x_{1:t}^a, z_{1:t}^b, x_0^b, x_{1:t}^b) p(x_{1:t}^a | z_{1:t}^a, u_{1:t}^a, x_0^a) p(x_{1:t}^b | z_{1:t}^b, u_{1:t}^b, x_0^b) \end{aligned} \quad (5.5)$$

$p(m_t | z_{1:t}^a, z_{1:t}^b, x_{1:t}^a, x_{1:t}^b, x_0^a, x_0^b)$ is the mapping problem with known poses and is calculated using ray tracing [3]. The other two terms which are posteriors of poses of each robot can be calculated using a particle filter.

The particle set for equation (5.5) is constructed as follows [28]. The i^{th} particle is given as

$$< x_t^{a(i)}, x_t^{b(i)}, m_t^{(i)}, w_t^{(i)} > \quad (5.6)$$

where $x_t^{a(i)}$ and $x_t^{b(i)}$ are poses of two robots at time t , $m_t^{(i)}$ is the map, and $w_t^{(i)}$ is the weight of the particle.

An important step in particle filtering is updating the weights of the particles. A recursive relation was presented in Chapter 3. The advantage of the recursive relation is that it allows us to perform full SLAM. This recursive relation is also used here. Since trajectories are independent and there is no line-of-sight observation (or it is ignored), the measurement model of each robot depends only on its own pose. Thus, weights are updated by the first measurement and then by the second measurement. In other words, by first observation, according to equation (3.66) presented in Chapter 3, weight of the i^{th} particle, $\check{w}_t^{(i)}$, is

$$\check{w}_t^{(i)} = p(z_t^a | x_t^{a(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}, \quad (5.7)$$

and by applying the second observation, the previous weight in equation (5.7) is

$$\begin{aligned} w_t^{(i)} &= p(z_t^b | x_t^{b(i)}, m_{t-1}^{(i)}) \check{w}_t^{(i)} \\ &\stackrel{\text{Eq. (5.7)}}{=} p(z_t^b | x_t^{b(i)}, m_{t-1}^{(i)}) p(z_t^a | x_t^{a(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}. \end{aligned} \quad (5.8)$$

Algorithm 5.2.1 summarizes the particle filter implementation for equation (5.5). In lines 4-5 predictions based on the motion models are performed. The sign \sim denotes that the samples on the left-hand side are generated according to the distribution given on the right-hand side. In line 6, weights of particles are updated based on equation (5.8). In line 7, the map of each particle is updated given the pose and measurement of each robot. The map is updated by ray tracing explained in [3], and function $\text{map}(\cdot)$ performs this operation. No matter how many pairs of poses and measurements are in the argument of the the function $\text{map}(\cdot)$, each pair is added to the map using ray tracing. In line 8, the updated particle is added to the posterior particle set. Finally, in lines 11-13, resampling is performed as explained in Chapter 3.

Algorithm 5.2.1 Particle filter for multiple robots with known relative poses and decoupled trajectories.

Input: set of particles representing posterior at time $t - 1$: S_{t-1} ,
control signals at time t : u_t^a, u_t^b ,
observations at time t : z_t^a, z_t^b .

Output: set of particles representing posterior at time t : S_t .

```

1:  $S_t \leftarrow \emptyset$ 
2: for  $i = 1 \rightarrow M$  do
3:    $< x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, m_{t-1}^{(i)}, w_{t-1}^{(i)} > \leftarrow S_{t-1}^{(i)}$ 
4:   sample  $x_t^{a(i)} \sim p(x_t^a | x_{t-1}^{a(i)}, u_t^a)$ 
5:   sample  $x_t^{b(i)} \sim p(x_t^b | x_{t-1}^{b(i)}, u_t^b)$ 
6:    $w_t^{(i)} \leftarrow p(z_t^a | x_t^{a(i)}, m_{t-1}^{(i)}) p(z_t^b | x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}$ 
7:    $m_t^{(i)} \leftarrow \text{map}(m_{t-1}^{(i)}, z_t^a, z_t^b, x_t^{a(i)}, x_t^{b(i)})$ 
8:    $S_t \leftarrow S_t \cup < x_t^{a(i)}, x_t^{b(i)}, m_t^{(i)}, w_t^{(i)} >$ 
9: end for
10: if  $N_{eff} < \frac{M}{2}$  then
11:   for  $i = 1 \rightarrow M$  do
12:     resample( $S_t$ )
13:   end for
14: end if
15: return  $S_t$ 
```

5.2.4.1.2 Coupled Trajectories

In multiple-robot SLAM, the direct relations between robots through line-of-sight observations and the indirect relations between robots through the environment, can be considered as secondary constraints for poses of robots. As mentioned, line-of-sight observations (direct encounters) make observations coupled. Moreover, observations of robots may also be coupled through collective observations (indirect encounters), when robots observe the same areas of the environment. Indirect and direct encounters may or may not happen at the same time, depending on the map of the environment. Taking into account these encounters can improve the mapping and localization process.

In this section, multiple-robot SLAM is extended to include line-of-sight and collective observations. Fig. 5.3 shows the Bayes net for two robots, a and b . In this figure,

solid black lines show the transition of the state of each robot. Dashed lines show the line-of-sight observations. Grey lines show the relation of the map and observations, where observations are again coupled through a common map.

To develop multiple-robot SLAM, it is desired to calculate the posterior over poses of all robots and the map. For two robots, the posterior is stated as

$$p(x_{1:t}^a, x_{1:t}^b, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b). \quad (5.9)$$

Following the discussion in Chapter 3, the posterior in (5.9) is represented as follows:

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) = \\ & p(m_t | z_{1:t}^a, z_{1:t}^b, x_{1:t}^a, x_{1:t}^b, x_0^a, x_0^b) p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \end{aligned} \quad (5.10)$$

Similar to the previous case, $p(m_t | z_{1:t}^a, z_{1:t}^b, x_{1:t}^a, x_{1:t}^b, x_0^a, x_0^b)$ is the mapping with known poses and is calculated using ray tracing [3]. However, the second term which is the joint posterior of poses of robots cannot be factored as in the previous case, because poses are correlated through encounters.

Prior to deriving the filtering for multiple robots with line-of-sight and collective observations, a few key relations are introduced in Sections 5.2.4.1.3 and 5.2.4.1.4. The purpose of these relations is to lay out a basis for multiple-robot SLAM based on Bayes filter. Then an implementation of the Bayes filter using the particle filter is presented in Section 5.2.4.1.5.

5.2.4.1.3 Joint Likelihood of Measurements

For two robots, the joint likelihood of measurements is defined as

$$p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}). \quad (5.11)$$

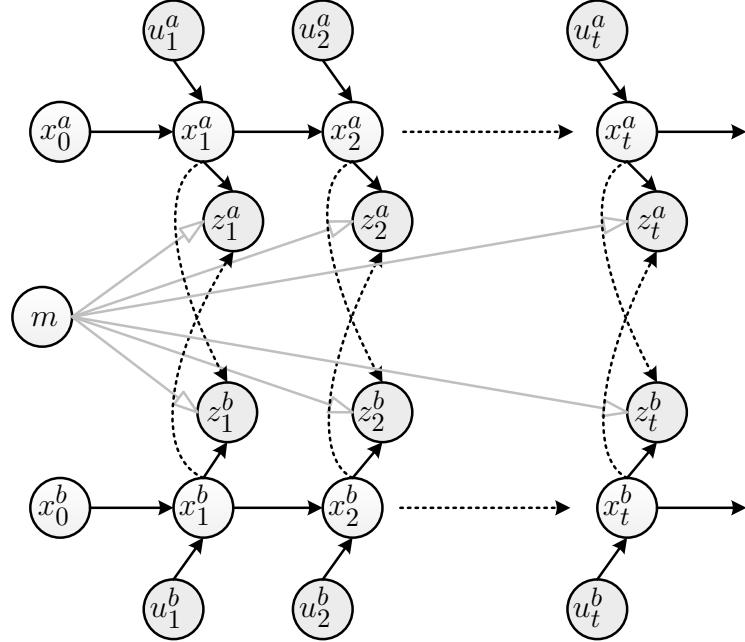


Figure 5.3: Bayes net for multiple-robot SLAM by two robots, a and b , with known relative poses and line-of-sight observation. Solid black lines correspond to the state transitions of each robot. Dashed lines correspond to the line-of-sight observations between robots and grey lines show the relation of the map and observations.

m_{t-1} is the map developed by the time $t - 1$. Notice that the likelihood is not conditioned on m_t . The reason for this is that m_t is constructed from measurements at time t , z_t^a and z_t^b ; therefore, conditioning z_t^a and z_t^b on m_t , causes the joint likelihood of z_t^a and z_t^b to be overconfident due to cyclic updates.

The joint likelihood represents the probability of observing measurements, z_t^a and z_t^b , given poses of the robots at time t and the map developed with all previous measurements. For this likelihood, a useful relation is extracted which is used in the development of the multiple-robot SLAM algorithm. This relation is stated in the following lemma.

Lemma 5.2.1. *For n robots, The joint likelihood of measurements can be represented as follows*

$$p(z_t^1, \dots, z_t^n | x_t^1, \dots, x_t^n, m_{t-1}) = \prod_{i=1}^n p(z_t^i | x_t^1, \dots, x_t^n, m_{t-1}). \quad (5.12)$$

This lemma is important because it allows the calculation of the joint likelihood of observations through likelihood of individual observations.

Proof. The proof of Lemma 5.2.1 is shown for two robots. For n robots the same approach can be used. It is shown that

$$p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) = p(z_t^a | x_t^a, x_t^b, m_{t-1}) p(z_t^b | x_t^a, x_t^b, m_{t-1}). \quad (5.13)$$

Consider Fig. 5.4. This figure depicts the Bayes net for two robots at time t . Observations at time t , z_t^a and z_t^b are coupled and dependent through two paths. One dependency path is through the poses with line-of-sight observations. This has been highlighted by the red path. For example, z_t^a depends on x_t^a . x_t^a has already been observed by z_t^b through the line-of-sight observation. z_t^b in turn depends on x_t^b . This circle of dependency implies that if x_t^a and x_t^b are known, then the path of dependency through the line-of-sight is no longer connected and z_t^a and z_t^b become conditionally independent. The other dependency path is through the map by observing the same areas of the map. This has been highlighted by the blue path. If the map is known, then z_t^a and z_t^b will no longer be coupled. Putting these two dependency paths together denotes that if the map and poses are known, then observations are conditionally independent.

□

5.2.4.1.4 Joint Posterior of Trajectories

In this section we investigate and analyze the joint posterior of trajectories. For two robots, the joint posterior of trajectories is defined as

$$p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b). \quad (5.14)$$

This joint posterior is used in multiple-robot SLAM which will be elaborated later. It represents the probability of all poses, given measurements and control actions. An

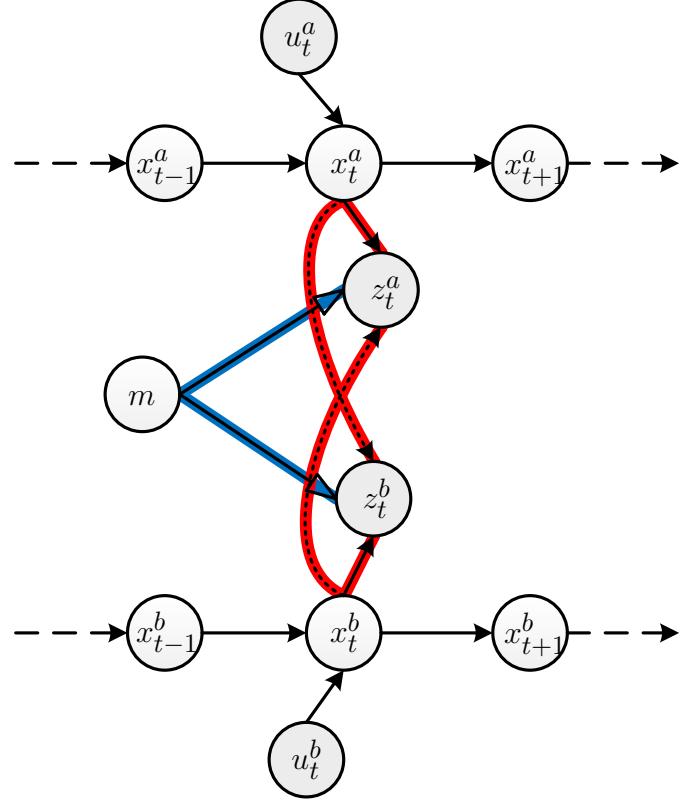


Figure 5.4: Bayes net for multiple-robot SLAM by two robots, a and b , with known relative poses and line-of-sight observation. Observations at time t , z_t^a and z_t^b , are coupled through two paths. One is through the poses with line-of-sight observation. The other is through the map by observing the same areas of map.

important relation is extracted which is used in the development of the multiple-robot SLAM algorithm. This relation is stated in the following lemma.

Lemma 5.2.2. *For n robots, The joint posterior of poses given all measurements and control actions can be represented recursively as follows*

$$\begin{aligned}
 & p(x_{1:t}^1, \dots, x_{1:t}^n | z_{1:t}^1, \dots, z_{1:t}^n, u_{1:t}^1, \dots, u_{1:t}^n, x_0^1, \dots, x_0^n) \\
 &= \eta p(z_t^1, \dots, z_t^n | x_t^1, \dots, x_t^n, m_{t-1}) p(x_t^1, \dots, x_t^n | x_{t-1}^1, \dots, x_{t-1}^n, u_t^1, \dots, u_t^n) \\
 &\quad \times p(x_{1:t-1}^1, \dots, x_{1:t-1}^n | z_{1:t-1}^1, \dots, z_{1:t-1}^n, u_{1:t-1}^1, \dots, u_{1:t-1}^n, x_0^1, \dots, x_0^n) \quad (5.15)
 \end{aligned}$$

where η is a normalizer.

This lemma is important for two reasons as follows:

- It is recursive which becomes very important in *full* SLAM, where not only current position of the robot but also the trajectory of the robot is desired.
- It includes joint likelihood of measurements conditioned on the poses and the map:

$$p(z_t^1, \dots, z_t^n | x_t^1, \dots, x_t^n, m_{t-1}). \quad (5.16)$$

The joint likelihood is already derived in Lemma 5.2.1.

Proof. The proof of Lemma 5.2.2 is shown for two robots, a and b . For n robots the same approach can be used. It is shown that

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &= \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) p(x_t^a, x_t^b | x_{t-1}^a, x_{t-1}^b, u_t^a, u_t^b) \\ &\times p(x_{1:t-1}^a, x_{1:t-1}^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b). \end{aligned} \quad (5.17)$$

Using the conditional probability rule, equation (3.4), the joint posterior is represented as (note: variables α_1 , α_2 , and α_3 are used just to show how the conditional probability is applied)

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &= p(\underbrace{x_{1:t}^a, x_{1:t}^b}_{\alpha_1} | \underbrace{z_t^a, z_t^b}_{\alpha_2}, \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3}) \\ &\stackrel{\text{Eq. (3.4)}}{=} \frac{p(\underbrace{z_t^a, z_t^b}_{\alpha_2} | \underbrace{x_{1:t}^a, x_{1:t}^b}_{\alpha_1}, \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3})}{p(\underbrace{z_t^a, z_t^b}_{\alpha_2} | \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3})} \\ &\times p(\underbrace{x_{1:t}^a, x_{1:t}^b}_{\alpha_1} | \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3}), \end{aligned} \quad (5.18)$$

the inverse of the probability in the denominator of equation (5.18) acts as a nor-

malizer; therefore, it has been replaced by a normalizer η .

$$\eta = \frac{1}{p(z_t^a, z_t^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)}. \quad (5.19)$$

The joint posterior is now presented as

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &= \eta p(z_t^a, z_t^b | x_{1:t}^a, x_{1:t}^b, z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &\times p(x_{1:t}^a, x_{1:t}^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \end{aligned} \quad (5.20)$$

The posterior can be written as:

$$\begin{aligned} & p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &= \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) \\ &\times p(x_{1:t}^a, x_{1:t}^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b), \end{aligned} \quad (5.21)$$

where in equation (5.20), in the following probability

$$p(z_t^a, z_t^b | x_{1:t}^a, x_{1:t}^b, z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b), \quad (5.22)$$

poses of the robots from time 1 to $t - 1$, $x_{1:t-1}^a$ and $x_{1:t-1}^b$, and observations from time 1 to $t - 1$, $z_{1:t-1}^a$ and $z_{1:t-1}^b$, are replaced with the map constructed by them, m_{t-1} . Note that in this distribution, the observations do not depend on the control actions.

By applying equation (3.9) to the second probability distribution in equation (5.21), using arbitrary parameters α_1 , α_2 , and α_3 to show how equation (3.9) is applied, the

posterior over the trajectory is

$$\begin{aligned}
& p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\
&= \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) \\
&\times p(\underbrace{x_{1:t-1}^a, x_{1:t-1}^b}_{\alpha_1}, \underbrace{x_t^a, x_t^b}_{\alpha_2} | \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3}) \tag{5.23}
\end{aligned}$$

$$\begin{aligned}
&\stackrel{Eq.(3.9)}{=} \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) \\
&\times p(\underbrace{x_t^a, x_t^b}_{\alpha_2} | \underbrace{x_{1:t-1}^a, x_{1:t-1}^b}_{\alpha_1}, \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b}_{\alpha_3}) \\
&\times p(\underbrace{x_{1:t-1}^a, x_{1:t-1}^b}_{\alpha_1} | \underbrace{z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b}_{\alpha_3}) \tag{5.24}
\end{aligned}$$

$$\begin{aligned}
&= \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) \\
&\times p(x_t^a, x_t^b | x_{1:t-1}^a, x_{1:t-1}^b, z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\
&\times p(x_{1:t-1}^a, x_{1:t-1}^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b) \tag{5.25}
\end{aligned}$$

$$\begin{aligned}
&\stackrel{Markov}{=} \eta p(z_t^a, z_t^b | x_t^a, x_t^b, m_{t-1}) \\
&\times p(x_t^a, x_t^b | x_{t-1}^a, x_{t-1}^b, u_t^a, u_t^b) \\
&\times p(x_{1:t-1}^a, x_{1:t-1}^b | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b) \tag{5.26}
\end{aligned}$$

□

5.2.4.1.5 Updating Weights of Particles

As mentioned, in particle filtering each particle holds an importance weight, denoting the similarity of the proposal distribution to the target distribution. It is defined as

$$w_t^{(i)} = \frac{p(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)}{\pi(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)}, \tag{5.27}$$

where $p(\cdot)$ is the target distribution and $\pi(\cdot)$ is the proposal distribution. As mentioned in Chapter 3, updating weights of particles is an important issue in the *full*

SLAM problem. Therefore, a systematic approach is required to update weights of particles. Please note that the definition of the weight as defined in equation (5.27) is in its general form and later the Markov assumption will be applied to it.

Theorem 5.2.1. *For n robots, assume the pose and observation of the j^{th} robot, $j = 1, \dots, n$, at time t is represented by x_t^j and z_t^j respectively. Assume that the map at time t is given by m_t and the weight of the i^{th} particle, $i = 1, \dots, N$, at time t is given by $\omega_t^{(i)}$. Then the weights of particles are updated through the following relation:*

$$w_t^{(i)} \propto \prod_{j=1}^n p(z_t^j | x_t^{1(i)}, \dots, x_t^{n(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}, \quad (5.28)$$

where the \propto shows the proportionality of the two sides of the relation.

Proof. The proof of the theorem is given for two robots. For n robots, the proof is similar. For two robots, a and b , it is shown that

$$w_t^{(i)} \propto p(z_t^a | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}. \quad (5.29)$$

To show equation (5.29), Lemmas 5.2.1 and 5.2.2 are used. By applying Lemma 5.2.2 and substituting the results from equation (5.26) in the numerator of equation (5.27) we will have

$$\begin{aligned} w_t^{(i)} &= \frac{p(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)}{\pi(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)} \\ &\stackrel{\text{Lemma 5.2.2}}{=} \eta p(z_t^a, z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(x_t^{a(i)}, x_t^{b(i)} | x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, u_t^a, u_t^b) \\ &\times \frac{p(x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)} | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b)}{\pi(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b)}. \end{aligned} \quad (5.30)$$

Using the Chain rule, the proposal distribution, $\pi(\cdot)$, is written as follows:

$$\begin{aligned} & \pi(x_{1:t}^{a(i)}, x_{1:t}^{b(i)} | z_{1:t}, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, x_0^b) \\ &= \pi(x_t^{a(i)}, x_t^{b(i)} | x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)}, z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b) \\ &\quad \times \pi(x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)} | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b). \end{aligned} \quad (5.31)$$

The weight updating in (5.30) becomes recursive:

$$\begin{aligned} w_t^{(i)} &= \frac{\eta p(z_t^a, z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(x_t^{a(i)}, x_t^{b(i)} | x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, u_t^a, u_t^b)}{\pi(x_t^{a(i)}, x_t^{b(i)} | x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)}, z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b)} \\ &\quad \times \frac{p(x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)} | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b)}{\pi(x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)} | z_{1:t-1}^a, z_{1:t-1}^b, u_{1:t-1}^a, u_{1:t-1}^b, x_0^a, x_0^b)} \end{aligned} \quad (5.32)$$

$$= \frac{\eta p(z_t^a, z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(x_t^{a(i)}, x_t^{b(i)} | x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, u_t^a, u_t^b)}{\pi(x_t^{a(i)}, x_t^{b(i)} | x_{1:t-1}^{a(i)}, x_{1:t-1}^{b(i)}, z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b)} w_{t-1}^{(i)}. \quad (5.33)$$

For each particle, by replacing the proposal distribution, $\pi(\cdot)$, in (5.33) with the motion model, $p(x_t^{a(i)}, x_t^{b(i)} | x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, u_t^a, u_t^b)$, the weight-update becomes

$$w_t^{(i)} = \eta p(z_t^a, z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}. \quad (5.34)$$

Using Lemma 5.2.1 and replacing the joint likelihood with individual likelihoods, the result is

$$\begin{aligned} w_t^{(i)} &= \eta p(z_t^a, z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)} \\ &\stackrel{\text{Lemma 5.2.1}}{=} \eta p(z_t^a | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)} \end{aligned} \quad (5.35)$$

$$\propto p(z_t^a | x_{t(i)}^a, x_t^{b(i)}, m_{t-1}^{(i)}) p(z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)} \quad (5.36)$$

□

Algorithm 5.2.2 explains the particle filter implementation for equation (5.10). Similar to the previous case, each particle holds trajectories of both robots, the map, and the weight associated with it. In line 4 prediction based on the motion model is performed. The key point of the algorithm is updating the weight which is done in line 5 based on Theorem 5.2.1. Notice that the measurements are conditioned on both poses. In line 6, the map of each particle is updated given the pose and measurement of each robot, similar to the previous algorithm. Finally, the updated particle is added to the particle set of the posterior. At the end, in lines 10-12, resampling is performed. Note that the main difference between this algorithm and the previous algorithm, Algorithm 5.2.1, is in the weight-update. In Algorithm 5.2.2, in line 5, where the weights of particles are updated, in each distribution, each measurement is conditioned on the poses of all robots. While in Algorithm 5.2.1, in line 6, in each distribution, each measurement has been conditioned only on the pose of the robot that makes the observation. In this section, this difference has been derived mathematically and it has been shown that the weight-update in Algorithm 5.2.2 takes into account the dependency of measurements on the poses of all robots. The implementation of the measurement models for Algorithm 5.2.2 has its own considerations which will be explained later. Note that the proposed algorithm is applicable to both feature-based and view-based maps. The choice of the map affects the detailed implementation of certain parts of the algorithm. For instance, if the map is a feature-based map and features are represented by Gaussian distributions, then updating maps (line 6) requires updating each feature through a Kalman filter [3]. But if the map is an occupancy grid map, developed by a range-bearing sensor, then maps can be updated using ray tracing [3].

Algorithm 5.2.2 Particle filter for multiple robots with known relative poses and line-of-sight observations.

$S_t = \text{multipleRobotSLAM}(S_{t-1}, u_t^a, u_t^b, z_t^a, z_t^b)$

Input: set of particles representing posterior at time $t - 1$: S_{t-1} ,
control signals at time t : u_{t-1}^a, u_{t-1}^b ,
observation at time t : z_t^a, z_t^b .

Output: set of particles representing posterior at time t : S_t .

```

1:  $S_t \leftarrow \emptyset$ 
2: for  $i = 1 \rightarrow M$  do
3:    $\langle x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, m_{t-1}^{(i)}, w_{t-1}^{(i)} \rangle \leftarrow S_{t-1}^{(i)}$ 
4:   sample  $\langle x_t^{a(i)}, x_t^{b(i)} \rangle \sim p(x_t^a, x_t^b | x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, u_t^a, u_t^b)$ 
5:    $w_t^{(i)} \leftarrow p(z_t^a | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) p(z_t^b | x_t^{a(i)}, x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}$ 
6:    $m_t^{(i)} \leftarrow \text{map}(m_{t-1}^{(i)}, z_t^a, x_t^{a(i)}, z_t^b, x_t^{b(i)})$ 
7:    $S_t \leftarrow S_t \cup \langle x_t^{a(i)}, x_t^{b(i)}, m_t^{(i)}, w_t^{(i)} \rangle$ 
8: end for
9: if  $N_{eff} < \frac{M}{2}$  then
10:   for  $i = 1 \rightarrow M$  do
11:     resample( $S_t$ )
12:   end for
13: end if
14: return  $S_t$ 

```

5.2.4.2 Unknown Relative Poses

In this section, the problem of the multiple-robot SLAM with unknown relative poses is studied. Results from the previous chapter are used to calculate the transformation; however, the problem of integrating the information from multiple sources is yet to be solved. Throughout the section, for simplicity, the problem is presented for two robots and when possible, it is explained for more than two robots.

Chronologically, when the relative poses are calculated on-the-fly, the SLAM problem for multiple robots can be divided into two parts: before the relative poses are known and after that. Before the relative poses are known, it is not possible to make a global map and have a joint posterior for poses because there is no common ground to connect the robots to each other. However, after the relative poses are known, the robots can share data and generate a global map. At the time that the relative poses

are known, a particle filter is constructed which proceeds to integrate all information until the end of the mapping. Before that time, robots were mapping and localizing individually and independently. Fig. 5.5 depicts this event for three robots, a , b , and c , where initially there is no knowledge about the relative poses of the robots. At time s , robots a and b know their relative poses. From this point on, a particle filter is constructed which includes poses of both robots and a map built by them. At time r , the relative pose of robot c is known by the other two robots. The particle filter is updated to include poses of all robots and the global map built by all of them. At each of these times, s and r , there are two types of updating tasks that should be performed. For example, for robots a and b , the information before and after time s should be integrated to the global frame. To integrate the information before time s , a novel method is proposed called *batch integration*. The information after time s is integrated using the particle filter proposed in Algorithm 5.2.2.

In this section, it is explained how once the realtime pose is known, the information before and after this specific time can be integrated. First it is explained how a particle filter is constructed when the robots find their relative poses. The particle filter is used to integrate the information from the time that the relative poses are known until the end of the mission. During this period, Algorithm 5.2.2, which performs multiple-robot SLAM with known poses, is used.

Then a solution is proposed to integrate the past information. The solution is called *batch integration*, and it also propagates the uncertainty of the relative transformation. The propagation should be performed on the future information and also on the past information. The batch integration is based on a novel uncertainty propagation called *particle regeneration*. This method of uncertainty propagation is integrated with a novel information integration method called *batch update*. In batch update, information, such as poses and maps, before the relative poses are known, is updated in batch mode.

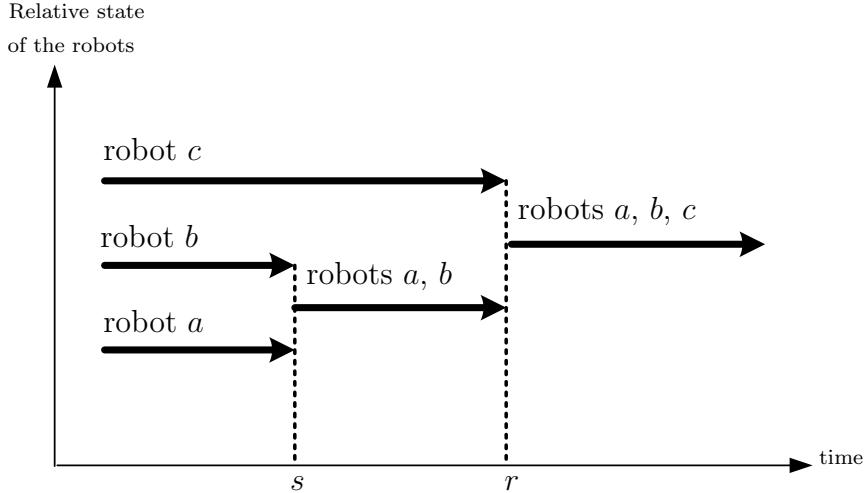


Figure 5.5: This diagram explains the relations between three robots, a , b , and c . Initially, robots do not know their relative poses. They start SLAM independently of each other.

5.2.4.2.1 Constructing Particle Filter

The transformations between the poses can be considered as variables in the state space of the filtering. Therefore as the mapping and localization proceed, the relative pose can also be updated resulting in more accurate mapping and localization. Assume that at time s ($1 < s < t$) the relative pose between two robots is somehow calculated. Then the multiple-robot SLAM problem can be stated as estimating the posterior over poses, the map (feature-based or view-based), and the transformation between poses:

$$p(x_{1:t}^a, x_{1:t}^b, \delta_{1:t}^{ab}, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, \delta_s^{ab}), \quad (5.37)$$

where δ_s^{ab} is the first relative transformation between the two robots a and b , known at time s and $\delta_{1:t}^{ab}$ shows the transformation from time 1 to time t . m_t is the desired global map built by robots at time t . Before time s , m_t does not exist; however, once the relative transformation is known at time s , m_t for all times, can be built as will be explained in the proposed solutions. Before time s , each robot maintains its own local map which is not shown in the distribution for the sake of brevity. Notice that

x_0^b is missing from the posterior. Using the Chain rule, equation (5.37) can be written as follows:

$$\begin{aligned}
& p(x_{1:t}^a, x_{1:t}^b, \delta_{1:t}^{ab}, m_t | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, \delta_s^{ab}) \\
&= p(m_t | x_{1:t}^a, x_{1:t}^b, \delta_{1:t}^{ab}, x_0^a, \delta_s^{ab}, z_{1:t}^a, z_{1:t}^b) \\
&\quad \times p(\delta_{1:t}^{ab} | x_{1:t}^a, x_{1:t}^b, x_0^a, \delta_s^{ab}, z_{1:t}^a, z_{1:t}^b) \\
&\quad \times p(x_{1:t}^a, x_{1:t}^b | z_{1:t}^a, z_{1:t}^b, u_{1:t}^a, u_{1:t}^b, x_0^a, \delta_s^{ab}). \tag{5.38}
\end{aligned}$$

The first probability in the right hand-side is the mapping with known poses [3]. The second and the third probabilities can be estimated using a particle filter with the i^{th} particle given as

$$< x_t^{a(i)}, x_t^{b(i)}, \delta_t^{ab(i)}, m_t^{(i)}, w_t^{(i)} >. \tag{5.39}$$

This particle formation models trajectories and the desired global map which are the goals of the multiple-robot SLAM algorithm. Note that if the map is known in advance, then there is no need to have the map in the particles, and in this case the problem is reduced to cooperative localization. This formulation solves for all variables; however, there is an important limitation with this solution and that is the scalability problem. As the number of the variables in the state space grows, complexity also increases which limits the algorithm to only a few robots. Therefore, this solution, although complete, has a major limitation in practice.

Instead of keeping a separate variable in the state space for the transformation and its uncertainty, they can be represented by poses indirectly. In fact, if the poses are known through the transformation, there is no need to have the transformation in the state space. The only major consideration is the propagation of the uncertainty of the transformation on the joint posterior of the poses and the map. This concern

can be taken care of by proper sampling of the particles which is explained in the next two subsections. Once the relative poses are known and the effect of uncertainty of the transformation is applied to them, the proposed solution in Algorithm 5.2.2, multiple robot SLAM with known poses, is used to update the maps and poses as observations are received.

5.2.4.2.2 Batch Integration

The batch integration is concerned with the integration of the information, maps and poses, from multiple robots prior to the time that the relative poses were known. In batch update, maps and poses are updated in batch mode. The batch mode means that the past data is not processed item by item; rather, maps and poses which were already processed and contain a lot of information, are updated in bulk. Batch update involves two main steps: first the uncertainty of the given relative transformation should be handled. Then the past information should be updated and integrated. The first step is addressed using the *particle regeneration* method. The second step is handled by the *batch update* method.

5.2.4.2.2.1 Particle Regeneration Assume that at time s the transformation between the two robots a and b is given with a Gaussian distribution as follows:

$$p(\delta_s^{ab}) = |2\pi\Sigma_s^{ab}|^{-\frac{1}{2}} \exp \left\{ \frac{-(\delta_s^{ab} - \mu_s^{ab})^T \Sigma_s^{ab} (\delta_s^{ab} - \mu_s^{ab})}{2} \right\} \sim \mathcal{N}(\delta_s^{ab}; \mu_s^{ab}, \Sigma_s^{ab}), \quad (5.40)$$

where $\mathcal{N}(\delta_s^{ab}; \mu_s^{ab}, \Sigma_s^{ab})$ denotes a Gaussian distribution with the mean of μ_s^{ab} and the covariance of Σ_s^{ab} defined as follows

$$\mu_s^{ab} = \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_\psi \end{bmatrix}, \quad \Sigma_s^{ab} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{x\psi}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 & \sigma_{y\psi}^2 \\ \sigma_{x\psi}^2 & \sigma_{y\psi}^2 & \sigma_{\psi\psi}^2 \end{bmatrix}. \quad (5.41)$$

μ_x and μ_y represent the translation and μ_ψ denotes the rotation. Generally, when a particle filter is constructed, the initial distribution, which particles are drawn from, is the belief that exists about the random variable. For example, the distribution can be Gaussian or even a delta function, where all particles have equal values and weights. The same principle applies for multiple robots performing SLAM. Without loss of generality, let us assume that the map and the poses of robot b will be transformed to the coordinates of robot a using the transformation in equation (5.40). At time s , a new particle filter is constructed which includes poses of both robots and a global map. (Note that the maps developed by each robot prior to time s will be used in a process called *batch update* to make a global map from the past information. The constructed particle filter will incrementally add the new measurements, received after time s , to this global map.) For the particle filter, the poses related to robot a can be selected from its last known pose. However, for robot b , the situation is different because its poses have to be transformed first. On the other hand, the transformation is uncertain and its effect on the pose and map should be taken into account. The proposed solution deals with this issue in three steps as follows.

- **Particle transformation:** All particles, representing the pose and map of robot b , are transformed according to the nonlinear transformation function.
- **Gaussian approximation:** Once each particle is transformed, the resulting pose of the transformed particle will have a nonlinear distribution. This nonlinear distribution is approximated with a Gaussian distribution. The motivation for the Gaussian approximation is to provide efficient and feasible computation for further processing. The main advantage of the Gaussian approximation is saving time.
- **Gaussian sampling:** From each Gaussian distribution assigned to the transformed pose, new particles are generated. These particles are used to update

maps and poses in the multiple-robot SLAM framework. Notice that each particle before the transformation holds a map. Once a particle is transformed, not only its pose is transformed, but its map is also transformed. The transformation of the map with uncertainty was already studied completely in the previous chapter. The transformed maps will be used as will be explained in the *batch update*.

These three steps have been illustrated in Fig. 5.6 and are explained in detail.

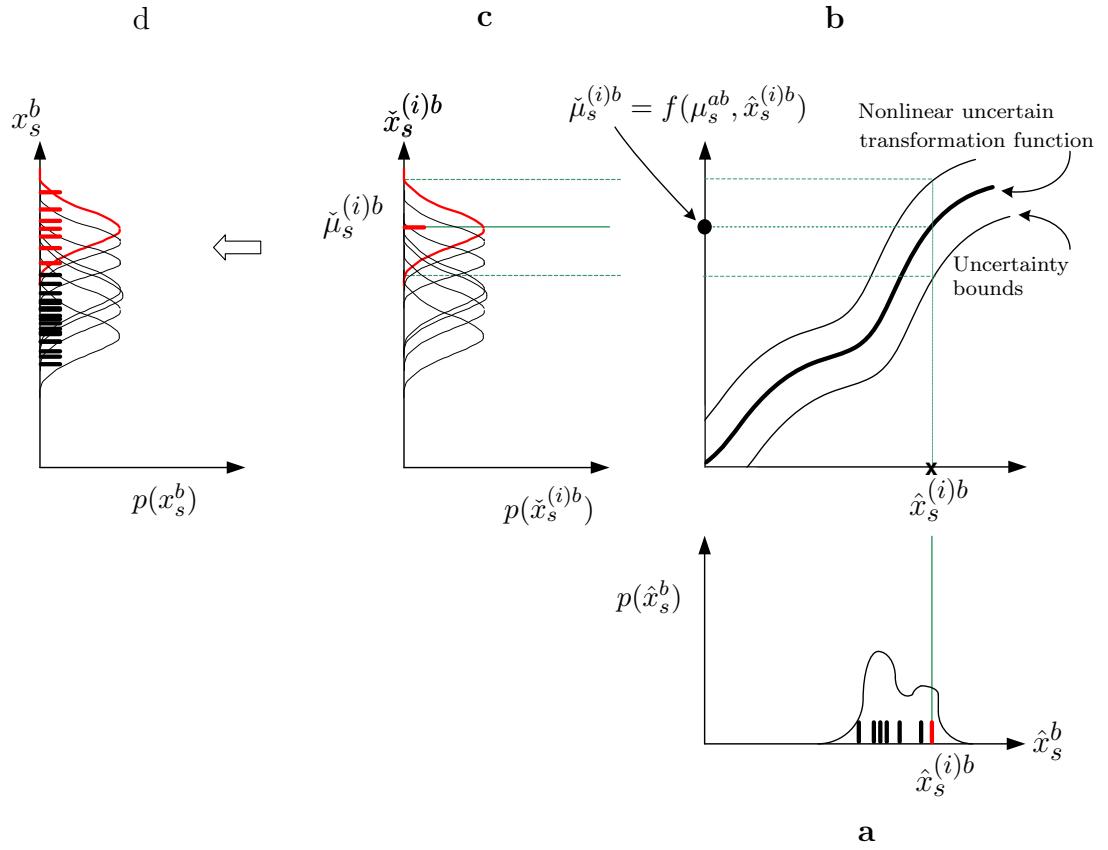


Figure 5.6: The proposed particle regeneration includes three steps: particle transformation, Gaussian approximation, and Gaussian sampling. **a)** The distribution of the pose of robots b before the transformation, at time s . Small bars are particles, modelling the distribution. A particle is highlighted to demonstrate the particle regeneration algorithm. **b)** Nonlinear and uncertain transformation function. **c)** All particles in **a** are mapped through the transformation function. The result of the transformation of each particle is approximated with a Gaussian distribution. **d)** From each Gaussian distribution, samples are generated to model it with particles.

Particle transformation:

At time s , let us assume that the distribution of the pose of robot b before the transformation is given by $p(\hat{x}_s^b)$. The distribution is modelled by \hat{n}^b particles. Fig. 5.6-a shows this distribution. The small bars show the particles representing the distribution. One of the particles is highlighted to show the effect of the uncertain transformation on it. Fig. 5.6-b depicts the uncertain and nonlinear transformation. The uncertainty bounds are also depicted in this figure. Once the i^{th} particle, $\hat{x}_s^{(i)b}$, is transformed by this uncertain transformation, the particle will also have an uncertain distribution (Fig. 5.6-c), which can be modelled by a Gaussian distribution.

Gaussian approximation:

Mathematically, the pose of each particle before the transformation, $\hat{x}_s^{(i)b}$, and after the transformation, $\check{x}_s^{(i)b}$, are related by the following function:

$$\check{x}_s^{(i)b} = f(\mu_s^{ab}, \hat{x}_s^{(i)b}) \quad (5.42)$$

where $f(\cdot)$ is the transformation function, defined as

$$\check{x}_s^{(i)b} = f(\mu_s^{ab}, \hat{x}_s^{(i)b}) \quad (5.43)$$

$$= \begin{bmatrix} \cos \mu_\psi & -\sin \mu_\psi & 0 \\ \sin \mu_\psi & \cos \mu_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \hat{x}_s^{(i)b} + \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_\psi \end{bmatrix}. \quad (5.44)$$

Once the i^{th} particle is transformed, its pose is modelled with a Gaussian distribution given as

$$p(\check{x}_s^{(i)b}) = |2\pi\check{\Sigma}_s^{(i)b}|^{-\frac{1}{2}} \exp \left\{ \frac{-(\check{x}_s^{(i)b} - \check{\mu}_s^{(i)b})^T \check{\Sigma}_s^{(i)b} (\check{x}_s^{(i)b} - \check{\mu}_s^{(i)b})}{2} \right\} \quad (5.45)$$

$$\sim \mathcal{N}(\check{x}_s^{(i)b}; \check{\mu}_s^{(i)b}, \check{\Sigma}_s^{(i)b}), \quad (5.46)$$

where the mean and covariance of the Gaussian is defined according to equation (5.44) as

$$\check{\mu}_s^{(i)b} = f(\mu_s^{ab}, \hat{x}_s^{(i)b}), \quad (5.47)$$

$$\check{\Sigma}_s^{(i)b} = F_\delta \Sigma_s^{ab} F_\delta'. \quad (5.48)$$

F_δ is the Jacobian of the transformation defined as

$$F_\delta = \frac{\partial f(\mu_s^{ab}, \hat{x}_s^{(i)b})}{\partial(\mu_s^{ab})}. \quad (5.49)$$

The Jacobian is calculated as

$$F_\delta = \begin{bmatrix} 1 & 0 & r_1 \\ 0 & 1 & r_2 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.50)$$

where, to simplify the representation, scalars r_1 and r_2 are defined as follows:

$$r_1 = [-\sin\mu_\psi \quad -\cos\mu_\psi \quad 0] \hat{x}_s^{(i)}, \quad (5.51)$$

$$r_2 = [\cos\mu_\psi \quad -\sin\mu_\psi \quad 0] \hat{x}_s^{(i)}. \quad (5.52)$$

Substituting the Jacobian in equation (5.49) yields the covariance of the transformed particle:

$$\begin{aligned} \check{\Sigma}_s^{(i)b} = & \\ & \begin{bmatrix} \sigma_{xx}^2 + 2\sigma_{x\psi}^2 r_1 + \sigma_{\psi\psi}^2 r_1^2 & \sigma_{xy}^2 + \sigma_{y\psi}^2 r_1 + \sigma_{x\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_1 r_2 & \sigma_{x\psi}^2 + \sigma_{\psi\psi}^2 r_1 \\ \sigma_{xy}^2 + \sigma_{y\psi}^2 r_1 + \sigma_{x\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_1 r_2 & \sigma_{yy}^2 + 2\sigma_{y\psi}^2 r_2 + \sigma_{\psi\psi}^2 r_2^2 & \sigma_{y\psi}^2 + \sigma_{\psi\psi}^2 r_2 \\ \sigma_{x\psi}^2 + \sigma_{\psi\psi}^2 r_1 & \sigma_{y\psi}^2 + \sigma_{\psi\psi}^2 r_2 & \sigma_{\psi\psi}^2 \end{bmatrix}. \end{aligned} \quad (5.53)$$

Therefore, once every particle is transformed, its pose will be represented by a Gaussian distribution, where the mean of the Gaussian distribution is defined according to equation (5.47) and its covariance is defined according to equation (5.53). Notice that with each particle, there is a map which is also affected by the uncertain transformation. The effect of the uncertain transformation on the maps was presented in the previous chapter.

Gaussian sampling:

When a particle filter is extended to include more state variables, the number of the particles must be increased to sufficiently represent the distribution of the new state. For example, if the pose and map of robot a and robot b are represented by 7 particles for each, then to represent the pose and map of both robots by a particle filter, a good approximation might be 50 particles. The reason for the increased number is that more particles are needed to approximate a bigger state vector.

Each of the transformed particles, $\hat{x}_s^{(i)b}$, should be regenerated to cover the bigger state space which is used in the particle filter initialized right after the moment that the relative poses are known. This is shown in Fig. 5.6-d. The transformed distribution shown in the figure is sampled to generate more particles. This sampling has two advantages. First, it represents the Gaussian distribution with particles which is required in particle filtering. Second, the particle set is increased as required for a bigger state space. Briefly, the red particle in 5.6-a, after the uncertain transformation, is represented by many red particles in 5.6-d. Each particle of each Gaussian distribution, shown in 5.6-c, holds a map, which is also affected by the transformation.

Algorithm 5.2.3 summarizes particle regeneration with its three steps: *particle transformation*, *Gaussian approximation*, and *Gaussian sampling*. The algorithm assumes that the information from robots b should be transformed into coordinates of robot a . The algorithm takes the particle set representing the pose and map of robots b before

the transformation is applied, \hat{S}_s^b , and outputs a particle set which represents the transformed state of the robot, S_s^b . In line 1, the set of output particles is initialized. In lines 2-3, the mean and covariance of the transformation matrix are assigned to the relevant parameters. In line 5, a loop starts to process all given particles. (\hat{n}_s^b is cardinality of \hat{S}_s^b .) First particle transformation and Gaussian approximation are performed. In line 7, for each particle, the mean of the transformed particle is calculated. In line 8, the covariance of each transformed particle is calculated. For this line, the result in equation (5.53) is used. In line 9, based on the calculated mean and covariance, a Gaussian distribution is assigned to model the pose of each transformed particle. In line 10, the `lup(·)` function returns the transformed map of the particle using the linearized uncertainty propagation, proposed in the previous chapter. For the pose of each transformed particle, there exists a Gaussian distribution, allowing Gaussian sampling to be performed. From each distribution, $\lfloor \frac{n}{\hat{n}_s^b} \rfloor$ poses are chosen and added to the output particle set with their corresponding maps (lines 11-14), where n is the desired population of the particles for multiple-robot SLAM. Notice that the maps for each Gaussian distribution are similar. Also, particle weights are not affected by the transformation. However, once the particles are regenerated, their weights have to be normalized. To normalize weights of particles, the weight of each particle is divided by the sum of weights of all particles.

5.2.4.2.2.2 Batch Update In the previous section, it is explained that at the moment the uncertain relative poses of the robots are known, particles holding maps and pose are transformed. So far, the poses of both robots are at global coordinates. Also, maps of both robots have been transformed. For both maps and poses, the uncertainty of the transformation has been propagated. Now the question is how all poses and maps in the global coordinates can be integrated to create a joint posterior and a global map for the past information. As mentioned, to answer this question,

Algorithm 5.2.3 Particle regeneration: $S_s^b = \text{particleRegeneration}(\hat{S}_s^b, \delta_s^{ab})$

Input: set of particles of posterior before the transformation at time s : \hat{S}_s^b ,
uncertain transformation between robots a and b at time s : δ_s^{ab} .

Output: set of particles of posterior after the transformation at time s : S_s^b .

```

1:  $S_s^b \leftarrow \emptyset$ 
2:  $\mu_s^{ab} \leftarrow \text{mean}(\delta_s^{ab})$ 
3:  $\Sigma_s^{ab} \leftarrow \text{cov}(\delta_s^{ab})$ 
4:  $\hat{n}^b \leftarrow \|\hat{S}_s^b\|$ 
5: for  $i = 1 \rightarrow \hat{n}^b$  do
6:    $\langle \hat{x}_s^{b(i)}, \hat{m}_s^{b(i)}, \hat{w}_s^{b(i)} \rangle \leftarrow \hat{S}_s^{b(i)}$ 
7:    $\check{\mu}_s^{(i)b} \leftarrow f(\mu_s^{ab}, \hat{x}_s^{(i)b})$ 
8:    $\check{\Sigma}_s^{(i)b} \leftarrow F_\delta \Sigma_s^{ab} F_\delta'$ 
9:    $p(\check{x}_s^{(i)b}) \sim \mathcal{N}(\check{x}_s^{(i)b}; \check{\mu}_s^{(i)b}, \check{\Sigma}_s^{(i)b})$ 
10:   $m_s^{b(i)} \leftarrow \text{lup}(\hat{m}_s^{b(i)}, \mu_s^{ab}, \Sigma_s^{ab})$ 
11:  for  $j = 1 \rightarrow [\frac{n}{\hat{n}^b}]$  do
12:    sample  $x_s^{(j)b} \sim p(\check{x}_s^{(i)b})$ 
13:     $S_s^b \leftarrow S_s^b \cup \langle x_s^{(j)b}, m_s^{b(i)}, \hat{w}_s^{b(i)} \rangle$ 
14:  end for
15: end for
16: return  $S_s^b$ 
```

batch update is used. Batch update allows us to integrate all past information, prior to the time that the relative poses are known, into the global coordinates without processing past raw sensor data items individually. The obvious advantage of the batch update is that it saves time by avoiding reprocessing past information.

Similar to the previous section, assume that particle set representing the state of robots a and b in the global frame at time s is given by S_s^a and S_s^b . First, these uncorrelated particles should be rearranged into a new particle set to represent the state for both robots. This is performed by uniformly drawing particles from both sets. Particle drawing must be without replacement, and continues until sets S_s^a and S_s^b become empty. Then maps of particles are fused to make a joint map from the past information. Algorithm 5.2.4 summarizes the batch update for two robots. The algorithm takes sets S_s^a and S_s^b as inputs. The cardinality of set S_s^a is n^a . But in Algorithm 5.2.3 the cardinality of set S_s^b was updated to be n particles, which is the

desired number of particles for the joint particle filter. The output of the algorithm is S_s^{ab} with cardinality of n . This set represents the joint posterior of the poses and map for both robots. In line 1, the output of the algorithm is initialized with an empty set. In line 2, the number of particles in the input set of S_s^a increased to the same number as in S_s^b . The desired number of particles should be n which is greater than n^a . In line 3, a loop starts to update all particles. In lines 4 and 5, particles without replacement are uniformly selected from the input sets. The function $\text{rand}(\cdot)$ does this operation. In these two lines, particles are chosen randomly and combined later to make a new set of particles. The motivation to perform random selection is to have diversity of the particles at the output of the algorithm, using the input particle sets. In line 6, For each pair of the selected particles, their maps are fused. For map fusion, the method based on the entropy filter, proposed in the previous chapter (Section 4.4.3.2.7), is used. In map fusion using the entropy filter, the additive property of the log odds representation of occupancy of the cells is utilized to reflect the uncertainty of the map cells. Note that if maps are feature-based maps, map fusion solutions such as [21] can be used. In line 7, weights of the particles are multiplied to generate the weight of the resulting particle. Finally, in line 8, the new particle is added to the output particle set. At the end, particle weights of the output set are normalized to sum to one. To normalize weights of particles, the weight of each particle is divided by the sum of weights of all particles.

Algorithm 5.2.5 summarizes the proposed solution, integrating all proposed algorithms. This algorithm can be thought of as a hybrid method, where map fusion results from the previous chapter are used to identify the relative poses and also a multiple-robot particle filter is used to integrate the information from multiple sources.

The algorithm runs single-robot SLAM if the relative poses are unknown. Otherwise it implements multiple-robot SLAM. The distinction between these are made by the

Algorithm 5.2.4 Batch update: $S_s^{ab} = \text{batchUpdate}(S_s^a, S_s^b)$

Input: set of particles representing posterior at time s : S_s^a ,
set of particles representing posterior at time s : S_s^b .

Output: set of particles representing joint posterior at time s : S_s^{ab} .

```
1:  $S_s^{ab} \leftarrow \emptyset$ 
2:  $S_s^a \leftarrow \text{populate}(S_s^a, n)$ 
3: for  $i = 1 \rightarrow n$  do
4:    $\langle x_s^{(i)a}, m_s^{(i)a}, w_s^{(i)a} \rangle \leftarrow \text{rand}(S_s^a)$ 
5:    $\langle x_s^{(i)b}, m_s^{(i)b}, w_s^{(i)b} \rangle \leftarrow \text{rand}(S_s^b)$ 
6:    $m_s^{(i)ab} \leftarrow \text{fuse}(m_s^{(i)a}, m_s^{(i)b})$ 
7:    $w_s^{(i)ab} \leftarrow w_s^{(i)a} w_s^{(i)b}$ 
8:    $S_s^{ab} \leftarrow S_s^{ab} \cup \langle x_s^{(i)a}, x_s^{(i)b}, m_s^{(i)ab}, w_s^{(i)ab} \rangle$ 
9: end for
10:  $S_s^{ab} \leftarrow \text{normalize}(S_s^{ab})$ 
11: return  $S_s^{ab}$ 
```

variable *mode*. If the relative pose is known, *mode* is *multipleRobotSLAM*; otherwise it is *singleRobotSLAM*. Inputs of the algorithm are control actions u_t^a and u_t^b , observations z_t^a and z_t^b , the previous state of the robot represented by particles P_{t-1} , and *mode*. The algorithm returns the new particle set P_t and the *mode*. In lines 2-4, the algorithm starts with single-robot SLAM as explained in Chapter 3. In line 5, maps are matched to check if it is possible to calculate the relative pose from maps. For this operation, any of the four solutions proposed in the previous chapter can be used. If it fails to find the relative pose (success = false), the state of each robot is returned and single-robot SLAM is implemented in the next iteration (line 7). If the relative pose can be calculated (success = true), then the relative pose and particle sets are used to implement the batch integration algorithm (lines 9-10), the mode is changed to *multipleRobotSLAM*, and a new particle set representing a state for multiple robots is returned. Once the mode is changed to *multipleRobotSLAM*, in lines 16-19, Algorithm 5.2.2 for multiple-robot SLAM with known poses is implemented.

Algorithm 5.2.5 Hybrid method:
$$(P_t, mode) = \text{hybrid}(u_t^a, u_t^b, z_t^a, z_t^b, P_{t-1}, mode)$$

```
1: if mode = singleRobotSLAM then
2:   ( $S_{t-1}^a, S_{t-1}^b$ )  $\leftarrow P_{t-1}$ 
3:    $S_t^a = \text{singleRobotSLAM}(S_{t-1}^a, u_t^a, z_t^a)$ 
4:    $S_t^b = \text{singleRobotSLAM}(S_{t-1}^b, u_t^b, z_t^b)$ 
5:    $(\delta_t^{ab}, success) \leftarrow \text{relativePose}(S_t^a, S_t^b)$ 
6:   if success = false then
7:      $P_t \leftarrow (S_t^a, S_t^b)$ 
8:   else
9:      $S_t^b = \text{particleRegeneration}(S_t^b, \delta_t^{ab})$ 
10:     $S_t^{ab} = \text{batchUpdate}(S_t^a, S_t^b)$ 
11:    mode = multipleRobotSLAM
12:     $P_t \leftarrow S_t^{ab}$ 
13:  end if
14:  return (P_t, mode)
15: else
16:    $S_{t-1}^{ab} \leftarrow P_{t-1}$ 
17:    $S_t^{ab} = \text{multipleRobotSLAM}(S_{t-1}^{ab}, u_t^a, u_t^b, z_t^a, z_t^b)$ 
18:    $P_t \leftarrow S_t^{ab}$ 
19:   return (P_t, mode)
20: end if
```

5.2.5 Advantages and Disadvantages

The proposed algorithm for multiple-robot SLAM started with the assumption that the relative poses were known. Then it was extended to include the general case, where the relative pose were unknown. To calculate the relative pose, results from the previous chapter were used and solutions were proposed to consider the effect of uncertainty on poses and maps.

In Algorithm 5.2.2, line-of-sight and collective observations were taken into account. This increases the accuracy of localization and mapping by providing more constraints on poses of the robots. The proposed batch integration method, consisting of Algorithm 5.2.3, particle regeneration and Algorithm 5.2.4, batch update, creates a filter once the relative poses are calculated and updates the maps and poses. It also takes into account the uncertainty of the relative poses and propagates it onto past and

future information. An obvious advantage of batch integration is that it saves time by avoiding reprocessing past data items. Also it requires less memory, since storing maps takes less space than storing raw data. In general, storing maps, whether feature maps, topological maps, or metric maps, requires less space than storing the raw data that the maps are built from. In fact the reduction in space comes at the cost of processing the raw data. A disadvantage of the method is that it lacks the flexibility to reprocess past data items, individually, which would make it possible to apply collective observations or redefine the past trajectories of the robots.

5.2.6 Discussion

This section discusses a few key issues in multiple-robot SLAM and the way they have been addressed in the proposed algorithm.

5.2.6.1 Effects of Communication Delays

As mentioned, out-of-sequence measurements are the ones that arrive at the destination in the wrong order. This can happen because of buffering or communication delays. Dealing with this problem is a major research field in communication systems, but here it is mentioned that a particle filter can get around this in certain circumstances.

Assume that the prediction rate in a SLAM algorithm is r_u Hz. Assume that the update rate is r_z Hz, where $r_z > r_u$. As an example, consider a robot with an encoder and a scanning laser rangefinder. The wheel encoder operates at 5 Hz resulting in predictions at the same rate and the laser ranger works at 40 Hz which updates the filter at this rate. In this example, for every prediction, there are 8 updates to be performed. Here it is shown that if $r_z > r_u$, then some of the out-of-sequence measurements can still be used when they arrive at the destination.

Lemma 5.2.3. *In between two consecutive predictions, measurements can be used to*

update the state of a particle filter in any order.

Proof. Assume that at time t , an update occurs. When the first measurement arrives at the destination, the weight of the i^{th} particle is updated as

$$\check{w}_t^{(i)} = p(z_t | x_t^{(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)} \quad (5.54)$$

The next time, when the second observation arrives, the weight becomes

$$\begin{aligned} w_{t+1}^{(i)} &= p(z_{t+1} | x_t^{(i)}, m_{t-1}^{(i)}) \check{w}_t^{(i)} \\ &\stackrel{\text{Eq. (5.54)}}{=} p(z_{t+1} | x_t^{(i)}, m_{t-1}^{(i)}) p(z_t | x_t^{(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}. \end{aligned} \quad (5.55)$$

Equation (5.55) shows that whether the distribution $p(z_{t+1} | x_t^{(i)}, m_{t-1}^{(i)})$ is processed first or the the distribution $p(z_t | x_t^{(i)}, m_{t-1}^{(i)})$, the weight of the particle will be the same. \square

The lemma is valid subject to the fact that the map for both measurements is the same. To have this condition, the update rate of the map should be less than the observation rate. In other words, maps should remain the same for all measurements in between the two predictions. In practice, and especially in occupancy grid maps, since updating maps is a time consuming process, the map update rate is on the order of 1 Hz. This lemma states that if a measurement arrives late, but before the next prediction, then it can still be used in the filtering process. In the next lemma, a relation is given to calculate the chance of usability of an out-of-sequence measurement in a particle filter.

Lemma 5.2.4. *Assume in a particle filter-based SLAM algorithm the prediction rate is r_u Hz and the update rate is r_z Hz, where $r_z > r_u$. If the maximum delay that an out-of-sequence measurement experiences to arrive at the destination is t_d , then the probability that the measurement can be used is $1 - t_d r_u$.*

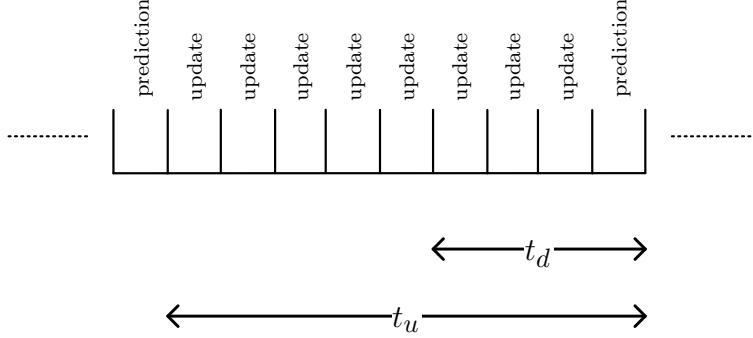


Figure 5.7: Sequence of updates and predictions in a particle filter. The rate of the update, r_z , is greater than the rate of the prediction, r_u . t_u is the time between two predictions. There are many updates between these two predictions. t_d is the delay which causes a measurement to be out-of-sequence. If the out-of-sequence measurement problem happens in the interval of $t_u - t_d$, then the delayed packet arrives at the destination before the next prediction, which makes it possible to use the measurement in the filter.

Proof. Fig. 5.7 shows an example of processing times. Time slots marked with “update” show the update instances and time slots marked with “prediction” show the prediction moments. t_u is the time between two predictions. If the out-of-sequence problem happens at the interval of $t_u - t_d$, then the measurement will arrive before the next prediction which means the measurement can still be used. Therefore, the probability that the out-of-sequence measurement can be used, p_m , is shown as

$$p_m = \frac{t_u - t_d}{t_u} = 1 - t_d r_u. \quad (5.56)$$

□

For instance, if the prediction and update rates are 5 Hz and 40 Hz, and t_d is 50 ms, the probability p_m is 75% which means only a quarter of the out-of-sequence measurements cannot be used in the filtering. The less frequent is the prediction compared with the update or the smaller is the t_d , the higher the probability p_m . Note that these

two lemmas are valid only in between two resampling steps. In practice, resampling occurs at least 10 times slower than the prediction.

5.2.6.2 Cyclic Update

Cyclic update occurs when a measurement is processed multiple times across the robots. This problem is related to the fact that information or data is communicated between the robots in the team. The main result of the cyclic update is the over-confidence in some variables that rely on repeated measurements. For example, if robots process and communicate their own poses and then receive the updated poses based on the same measurements, a cyclic update happens. A solution to prevent cyclic updating is to avoid or minimize sharing state variables. For example, sharing and communicating poses or maps for multiple times might cause this problem. In the proposed algorithm, raw data was constantly communicated and only once were the maps used to find the relative poses. This instance happens in Algorithm 5.2.4, line 6, where maps are used to find the relative transformations. Once the relative transformations are known, maps will not be used any more and the risk of recurring cyclic updates is minimized. Once the relative poses are known, particle filtering continues with the processing of raw data, such as laser scans or odometry, received from all robots.

5.2.6.3 Closing Loops

When a robot moves thorough an unknown environment and at some point observes areas which have not been observed for a long time, loop closure happens. Loop closure is very important for consistent mapping and localization. Different algorithms deal with this problem by different means. While in EKF or GraphSLAM it is handled by the covariance matrix, in particle filtering, it is addressed by the diversity of the particles. In other words, since every particle holds an estimate of the trajectory

and the map through that trajectory, then there are multiple hypotheses representing the map and the trajectory of the robot. When loop closure happens, having this diversity enables the robots to identify it correctly. Having more particles means having more hypotheses which enables the robots to identify loop closures efficiently and accurately. Each particle holds trajectories of all robots, which means a diverse population of all trajectories by all robots is maintained by the particle filter. Thus, whenever line-of-sight or collective observations are made, loop closure can happen.

5.2.6.4 Updating Maps and Poses

In multiple-robot SLAM, once robots in the team have a common reference to integrate their information, the problem of updating past and future maps and poses emerges. This is in direct relation to loop closure, because once the relative poses between the robots are known and past information has been updated properly, then loop closure might happen right away. In the proposed algorithm, the batch integration method updates maps and poses prior to the time that the relative poses are known. Maps are updated through the entropy filter (Section 4.4.3.2.7), and poses are updated by transforming them through the uncertain transfer function. Although it is more flexible to reprocess that past data items individually; however, batch update saves the reprocessing time and limits the memory usage by storing processed information rather than raw data.

5.2.7 Contribution

The contributions of the current work include a few novel improvements in different aspects of multiple-robot SLAM. These improvements are multiple-robot SLAM with coupled trajectories, considering uncertainty of the relative transformation between robots, updating maps and poses, and integrating map matching with particle filtering.

- **Multiple robots with line-of-sight observations:** The proposed method in Algorithm 5.2.2 includes line-of-sight observations between robots to improve the performance of the SLAM. Unlike most other works, in this work poses of robots are considered to be correlated and this can significantly affect the robustness and reliability of multiple-robot SLAM.
- **Particle regeneration:** Uncertainty of the transformation is usually ignored in the cooperative mapping and localization algorithms. This can affect the confidence in maps and poses. The proposed particle regeneration method in Algorithm 5.2.3 takes care of the uncertainty of the transformation during the mapping and localization process.
- **Batch update:** This method, proposed in Algorithm 5.2.4, allows us to update past maps and poses, prior to the time that the relative poses are known, by processing them in batch, rather than reprocessing individual data items one by one. The obvious benefit of this algorithm is saving time by avoiding reprocessing past data and also saving memory by not storing all past raw data.
- **Hybrid method:** The proposed method in Algorithm 5.2.5 deals with the problem that if the transformation between robots is not known then it has to be extracted from other sources of information such as maps or line-of-sight observations. In general, it is not guaranteed that robots will meet each other at a point to calculate the relative transformation from line-of-sight. However, by processing maps this problem can be solved. Previous map merging solutions are integrated with the current work to make it a practical and operational solution. By adding map merging, the proposed solution becomes a hybrid solution where maps and raw data are used to build and maintain a global map.

5.2.8 Experiment

In this section, two simulated experiments are performed in MATLAB. The first experiment demonstrates the effect of the line-of-sight observations on reducing the error of the localization and mapping, assuming that the relative poses of the robots are known. In the second experiment, the proposed algorithm for unknown relative poses is implemented. Both experiments are performed for feature-based SLAM; however, all algorithms in this chapter are also applicable to view-based SLAM.

5.2.8.1 Case1: Line-of-sight Observations

The first experiment describes the effect of the line-of-sight observations on efficiency of multiple-robot SLAM. To evaluate the effect of line-of-sight observations, the results of multiple-robot SLAM with and without line-of-sight observations are compared. For this experiment, it is assumed that the relative poses of the robots are known. The simulated robots, observations, and control actions are developed in MATLAB. Fig. 5.8 shows the simulated world. Two robots are used for this simulation, depicted by the red triangles. The blue path is the trajectory along which the robots move. The green stars are landmarks to be detected by the robots. The observations between the robots and the landmarks are shown in black. The line-of-sight observations between the robots are shown in red.

Probabilistic models, including motion and observation models, are the core components of the algorithm. The motion model and observation models for feature-based SLAM are well-presented in [3] by Thrun *et al.* The line-of-sight observation model is similar to the feature observation model of each robot, except that the line-of-sight observation, which includes range and bearing, is made from another robot rather than from a feature. A similar concept applies to view-based SLAM.

Fig. 5.9 shows the mean squared error of the position of the landmarks. The robots move for 400 s, each making one loop following the given path. The error when the

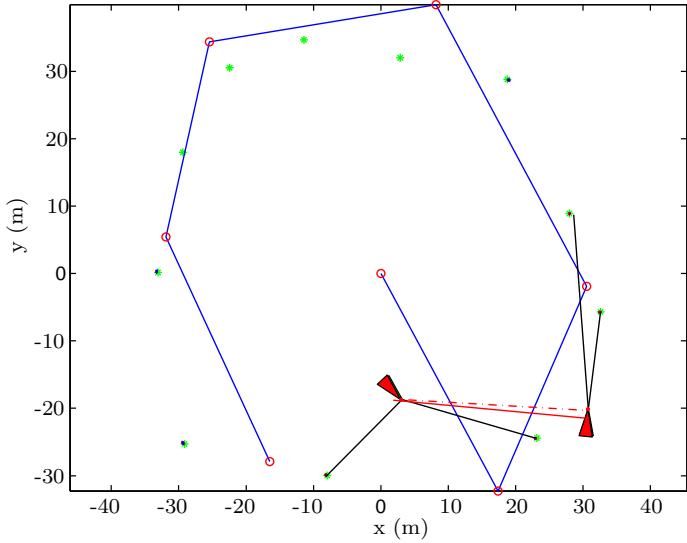


Figure 5.8: The simulated world. Two robots are depicted by the red triangles. The blue path is the trajectory that the robots move along. The green stars are landmarks to be detected by the robots. The observations between the robots and the landmarks are shown in black. The line-of-sight observations between the robots are shown in red.

line-of-sight observations are ignored is shown in black. The error when the line-of-sight observations are taken into account, using the proposed algorithm, is depicted in red. As the results show, with line-of-sight observations, the error is lower, which supports the claim made earlier. In steady state, which happens approximately at $t = 150$ s, the difference between the errors of the two scenarios is approximately 0.5 m which shows a significant improvement.

5.2.8.2 Case2: Unknown Relative Poses

The second experiment describes the hybrid multiple-robot SLAM algorithm. This algorithm is applicable to both view-based and feature-based SLAM; however, in this experiment, only its implementation for feature-based SLAM is discussed. In this experiment, it is assumed that the relative transformation between the robots is not known in advance.

As in the previous experiment, the simulated robots, observations, and control actions

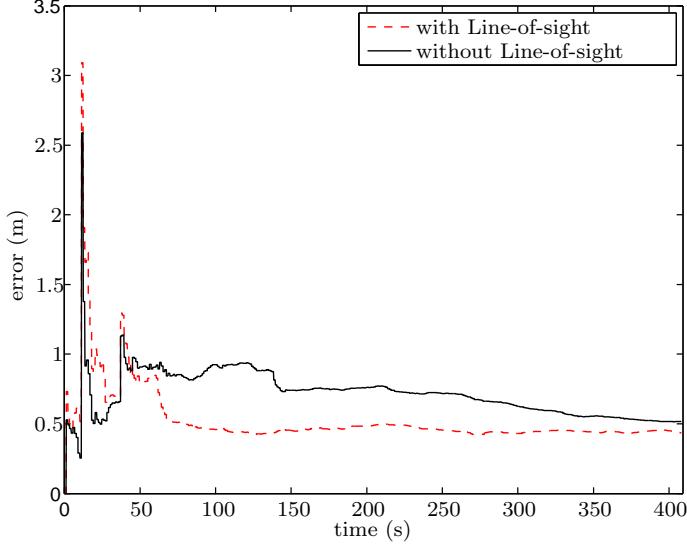


Figure 5.9: The mean squared error of the position of the landmarks. The error when the line-of-sight observations are ignored is shown in black. The error when the line-of-sight observations are taken into account, using the proposed algorithm, is depicted in red.

are developed in MATLAB. Two robots are used for this simulation. Also, it is assumed that the relative transformation between the robots is not known, and it is calculated from the map.

Based on Algorithm 5.2.5, the robots start in the *singleRobotSLAM* mode. Periodically, their maps are matched, following line 5 in Algorithm 5.2.5. For the implementation of the function *relativePose*(\cdot), the iterative closest point (ICP) algorithm is used, in which features are considered as points to be matched. Once the matching is acceptable based on the error of the matching, the relative transformation between the maps is used as the relative transformation between the robots, and the mode is switched to *multipleRobotSLAM*. New particles are generated for this mode, and the robots continue to perform multiple-robot SLAM with known relative poses, as explained in the previous experiment. In the following paragraphs, detailed analyses of important issues such as the uncertain transformation and map merging in feature-based maps are introduced.

5.2.8.2.1 Transforming Features with an Uncertain Transformation

In Section 4.4.3.1, transforming metric maps with an uncertain transformation was studied. For feature-based maps, the same formulation applies, except that for feature maps only certain parts of the proposed solution are required which will be explained by an example.

Fig. 5.10 shows the effect of an uncertain transformation on features. It is assumed that the features are translated by [3, 2] along X and Y axes and rotated 15°. The transformation is uncertain and the covariance of the transformation, as defined in (5.37), is given as follows:

$$\Sigma_{tr} = \begin{bmatrix} 0.1 & 0.02 & 0.01 \\ 0.02 & 0.15 & 0.01 \\ 0.01 & 0.01 & 0.099 \end{bmatrix}. \quad (5.57)$$

Fig. 5.10-a and Fig. 5.10-b show the result of the transformation when the uncertainty of the feature, q , before the transformation is zero. The covariance of the transformed point, Σ_r , is calculated using the following relation, which was derived in Section 4.4:

$$\Sigma_r = F_{xy\psi} \Sigma_{tr} F_{xy\psi}^T + F_q \Sigma_q F_q^T. \quad (5.58)$$

where $F_{xy\psi}$ and F_q are the Jacobians of the transformation function, $f(\cdot)$ as defined in (5.35) , and are given based on:

$$F_{xy\psi} = \frac{\partial f(\psi, x, y, q)}{\partial(\psi, x, y)}, \quad F_q = \frac{\partial f(\psi, x, y, q)}{\partial(q)}. \quad (5.59)$$

Σ_q is the covariance of the point q which in Fig. 5.10-a and Fig. 5.10-b is zero, meaning that the feature before the transformation has no uncertainty. In Fig. 5.10-b, the feature is in a different location, compared with Fig. 5.10-a. According to equation (5.58), the uncertainty of the transformed point is a function of the position

of the feature point, q . Therefore, points at different locations will experience different uncertainties after the transformation.

Fig. 5.10-c and Fig. 5.10-d show the result of the uncertain transformation, when the uncertainty of the feature before the transformation is not zero. For this case, the uncertainty of the features before the transformation is

$$\Sigma_q = \begin{bmatrix} 0.2 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}. \quad (5.60)$$

Similarly, in Fig. 5.10-d, the feature is in a different location, compared with Fig. 5.10-c. Since the uncertainty of the transformed point is a function of the position of the feature point, points at different locations with similar covariance matrices before the transformation will experience different uncertainties after the transformation. Note that the uncertainty of the transformed point in Fig. 5.10-d is larger than the one in Fig. 5.10-b, because in Fig. 5.10-d, the initial uncertainty of the point before the transformation is not zero.

5.2.8.2.2 Transforming Poses with an Uncertain Transformation

Once the relative transformation between the robots is known, not only features but also poses of the robots should be transformed. This is required to perform multiple-robot SLAM with all agents. In equation (5.53), a relation was presented that calculates the covariance of the pose of a robot after transformation. Here this relation is explained with an example. Fig. 5.11 shows a robot in a black triangle, located at [2, 2] with an initial orientation of 30° . The pose of the robot is translated by [3, 2] along X and Y directions and rotated 15° . The transformation is uncertain and its covariance is given in equation (5.57). The covariance of the transformed pose is calculated using equation (5.53). Since it is not easy to visualize the covariance ellipse for position and orientation of the transformed robot, 100 pose samples are

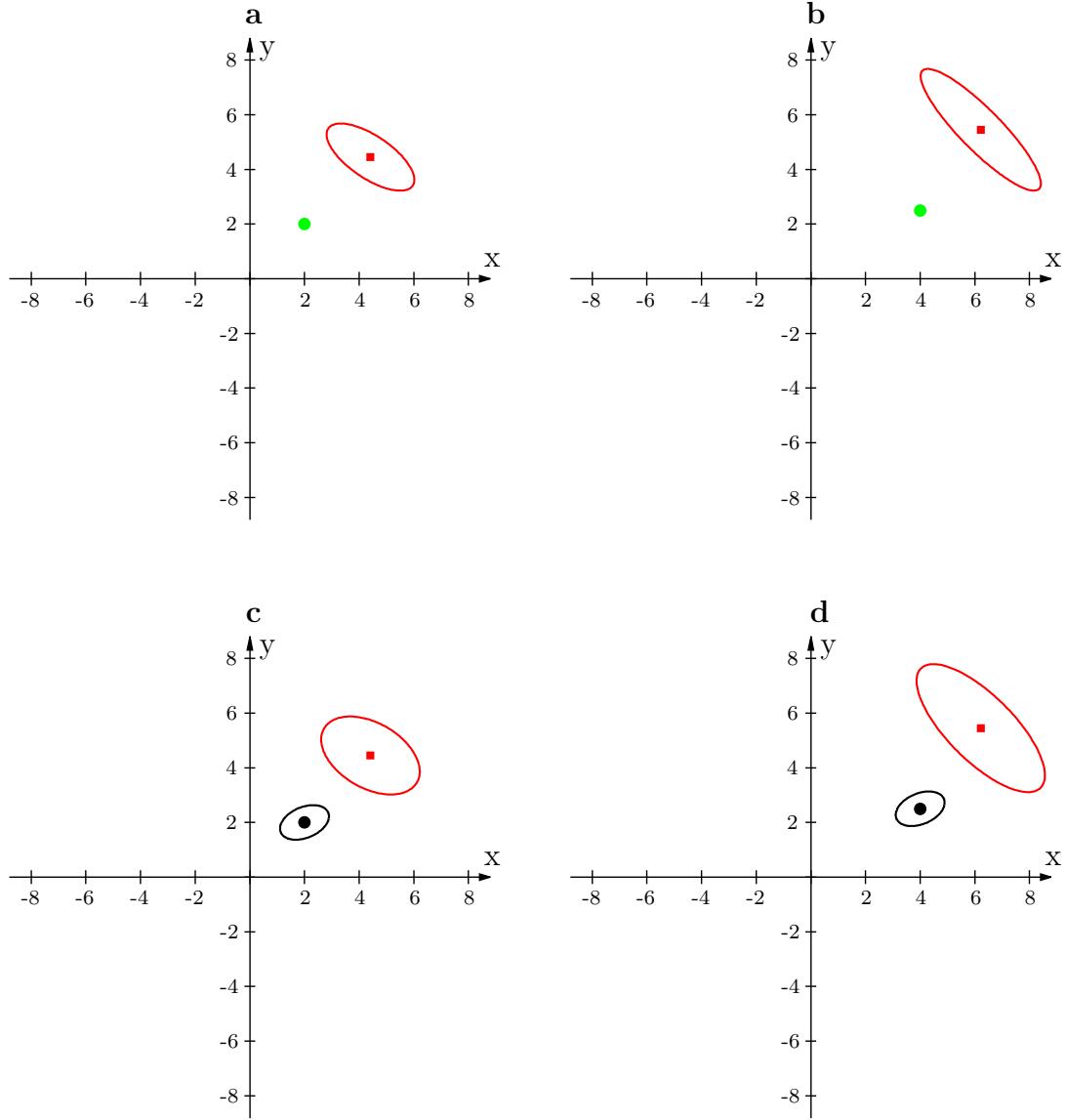


Figure 5.10: Transformation of a feature. Green points show features with no position uncertainty. Black points show uncertain features with their position uncertainty ellipse. Red squares demonstrate the transformed features. For all examples, the translation is [3, 2] and the rotation is 15° . The transformation for all examples is uncertain. **a)** A deterministic feature is transformed. **a)** Another deterministic feature, which is far from the center, is transformed. **c)** An uncertain feature is transformed. **d)** Another uncertain feature, which is far from the center, is transformed.

generated from the calculated covariance and illustrated by red triangles. The red dotted ellipse shows only the covariance of the position of the robot. Note that initial uncertainty of the pose of the robot before the transformation is taken care of by the diversity of the particles and this figure represents the pose of only one particle.

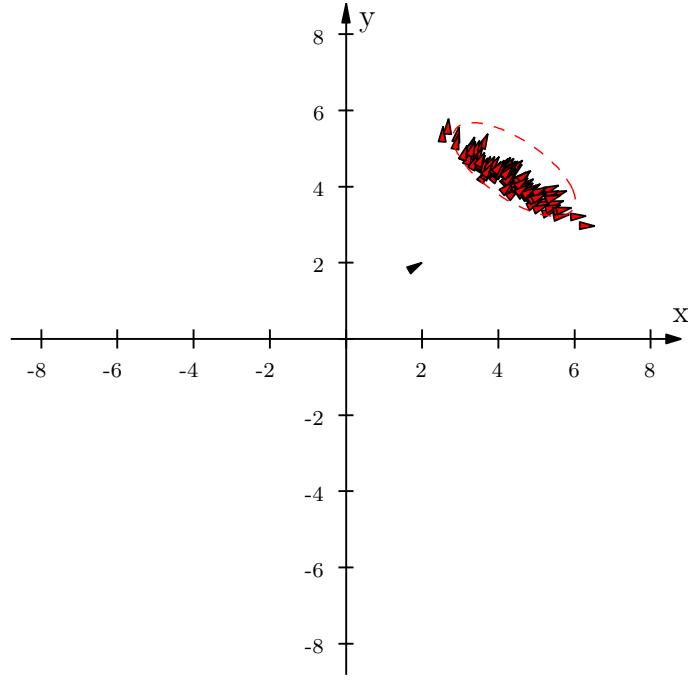


Figure 5.11: Uncertain transformation of the pose of a robot. To visualize the uncertainty of the transformed pose, 100 pose samples resulting from the uncertain transformation are shown.

5.2.8.2.3 Merging Feature Maps

Once two feature maps are aligned by a transformation, they must be merged to generate a global map. Merging maps can be done by copying all features from one map to another; however, due to noise and uncertainty of the position of the features, duplicate features might exist. These duplicates should be merged to avoid incorrect mapping. To do this, if the distance between two features from aligned maps is less than a predefined threshold, then they should be merged and considered as one feature. This has been shown for two features in Fig. 5.12. In Fig. 5.12-a, two features are depicted. One of them is shown by a circle, the other one by a diamond. Assume the diamond feature has been transformed from another map and the circle one was already existing in the current map. For each pair of such features, their distance is calculated. If the distance is less than the threshold, then the features are averaged to generate a new feature which represents both features. Fig. 5.12-b shows

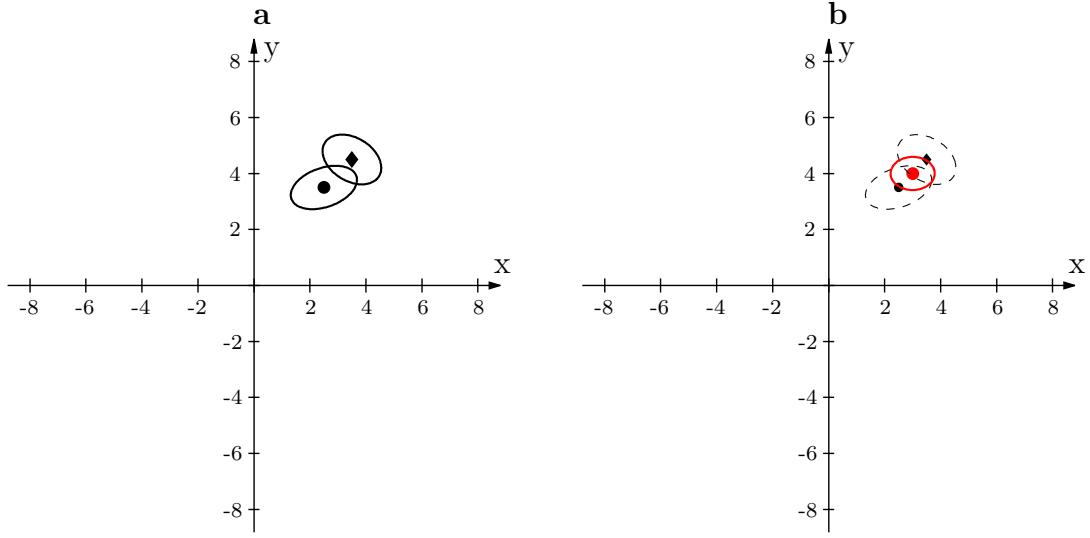


Figure 5.12: Merging two features. **a)** Two features whose separation is smaller than a given threshold. **b)** The new merged feature is shown in red. Features are merged by averaging their mean positions. Their covariances are also linearly added.

the new feature and its covariance ellipse in red. Note that since averaging is a linear operation, the covariance of the merged point is the result of the linear summation of the covariances of the points before merging.

5.2.8.2.4 Results

Fig. 5.13-a shows the environment in which the proposed hybrid algorithm has been tested. Two robots start from different positions. Without loss of generality, the origin of the first robot is assumed to be the origin of the global coordinates. Each robot is represented by two triangles: a green triangle which shows the actual pose of the robot and a red triangle which shows the estimated pose of the robot. For the first robot, both triangles are located at the origin. For the second robot, the actual position is at [15, -10] with the orientation of 0°. The estimate of this robot is located at the origin. Before the relative transformation is known, each robot develops its own map. The maps are overlaid in the global frame, without being aligned. Fig. 5.13-b shows the simulation at time $t = 27$ s. All features identified by the second robot are misplaced (shown by black arrows) because the relative

transformation is not known. Note that for the same reason, the estimated pose of the second robot in the global coordinates, shown by a red triangle, is different from its actual pose in the global frame (shown by a red arrow). As mentioned, after each measurement update, map matching is performed. At time $t = 28$ s, map matching is able to calculate the required transformation for the maps. The calculated translation is $[15.26, -10.18]$ and the orientation is 0.81° . At this point the map and the pose of the second robot are transformed to the global coordinates, maps are merged, and particles are regenerated to perform multiple-robot SLAM, as was demonstrated in the previous experiment.

The duration of the experiment, performing two loops in the environment, was 869 s. 30 particles were used for each robot when they were doing single-robot SLAM and 100 particles for both robots when doing multiple-robot SLAM. Fig. 5.13-c shows the error of the localization of the map's features for the duration of the experiment.

5.2.9 Summary

In this section a general method was proposed to perform SLAM with multiple robots. The proposed method started with the assumption that relative poses are known. In this case line-of-sight observations were taken into account. Then the problem was extended to include unknown relative poses. To calculate the relative pose, map merging was used. Once the relative poses are known, the uncertainty of the transformation, representing the relative poses, were propagated to the past and future maps and poses. Moreover, past information was updated in batch to save time and memory. Other issues in multiple-robot SLAM, such as out-of-sequence measurements, cyclic update , and loop closure, were also discussed and reviewed.

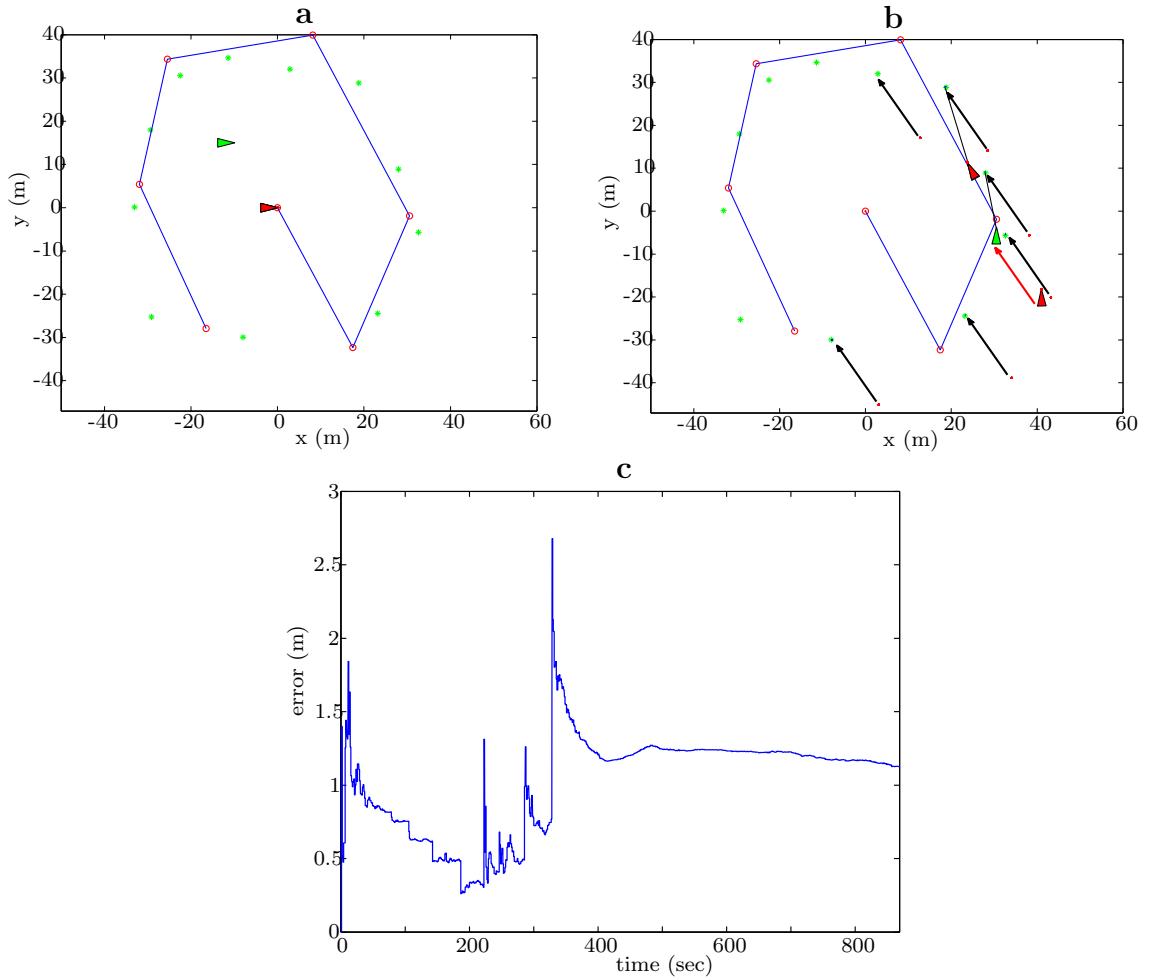


Figure 5.13: The hybrid SLAM algorithm. **a)** The test environments with two robots. **b)** Maps and robots before the relative poses are known. **c** The mean squared error of the position of the landmarks.

Part III

Rotorcraft SLAM

In recent years, researchers have intensively worked on the development of flying machines capable of performing complicated missions with little human supervision. These vehicles are commonly known as unmanned aerial vehicles (UAVs). UAVs have no crew onboard and are flown either remotely by a pilot at a ground control station or autonomously through a pre-planned program. Rotorcraft, vehicles with multiple rotors, have attracted more attention recently, with quadrotors being the most popular ones. These multiple-rotor vehicles, also known as flying robots, have outstanding manoeuvrability. Furthermore, they are less complex compared with other rotorcraft such as helicopters. These advantages have made quadrotors suitable for civil and military applications.

With the main objective of developing efficient quadrotors, capable of accomplishing complicated tasks autonomously, a significant amount of work has been dedicated to designing and building efficient sensing suits to perceive the environment and estimate the state of the vehicle. Knowledge of the surrounding world and variables representing the vehicle in the world are used to stabilize the vehicle. Once the vehicle is stable, it can perform more difficult and complicated tasks autonomously.

Early prototypes of UAVs were dependent on inertial dead reckoning aided by global positioning system (GPS). However, GPS signals have poor performance in urban settings and inclement weather conditions. Moreover, there is no GPS reception in most indoor environments and enclosed spaces which are referred to as GPS-denied environments. To obtain position and orientation estimates in these environments, an alternative solution is to equip vehicles with inertial, optical, and ranging sensors. This part of the thesis studies the possibility of using state-of-the-art sensing technologies to perform perception in GPS-denied environments and achieve autonomy with minimum human intervention. The proposed work is presented in two chapters: Chapter 6 presents the background information, and Chapter 7 introduces the proposed solution for perception and navigation of an autonomous quadrotor.

Chapter 6

Background

This chapter presents background to rotorcraft perception. In Section 6.1, some information about small unmanned flying robots, quadrotors, is presented. Section 6.2 presents background to navigation which includes control, state estimation, path planning, localization, mapping, and high level mission planning . Finally, Section 6.3 summarizes the chapter.

6.1 Quadrotor

A quadrotor, also called quad rotorcraft, is a rotorcraft which is lifted, controlled, and propelled by four rotors. The quadrotor can take off and land vertically. It can fly in any direction without changing its heading. This section presents background to its history, motivation, definition, advantages, and disadvantages.

6.1.1 History

The history of the quadrotor is the story of the evolution of a simple idea: unlimited flight in all directions. The realization of the idea started with one rotor. The first known man-made flying device is the Chinese top (Fig. 6.1-a). The earliest versions

of the Chinese top, which dates to 2500 years ago, consisted of feathers at the end of a stick. By rapidly spinning the stick between the hands, a lift force was generated. Then by releasing the stick, the toy was free to fly [148]. The famous Helix, by Leonardo da Vinci (1452-1519), is the oldest known design of a controlled flying machine (Fig. 6.1-b). The Helix was a screw-shaped airfoil attached to a wooden framework. The airfoil was spun by a pilot to generate thrust. However, this idea was not feasible because high muscle power was required to spin the airfoil [148]. Since then, many passionate researchers have attempted to design and build a flying rotorcraft, but the first successful flight was performed on September 29, 1907, by two French brothers, Jacques and Louis Breguet. Their rotorcraft, Gyroplane No.I, shown in Fig. 6.1-c, could lift up its pilot about 0.6 meters for one minute. Their design had four rotary wings and it was neither controllable nor steerable. During their flight, four men controlled the machine from the ground. Later in 1922, the design was improved by Dr. George de Bothezat and Ivan Jerome [149]. They designed an X-shaped structure shown in Fig. 6.1-d. It weighed 1678 kg with four arms, each nine meters. Each arm had an 8.1 meters diameter six-blade rotor. Because of its size and mechanical complexity, it suffered from poor performance and could never hover more than five meters.

Recently, advances in processing and sensing technologies have led to better and higher performance designs with different scales and applications such as the Bell Boeing Quad TiltRotor, ArduCopter [152], Parrot AR.Drone [153], Aeryon Scout [154], Draganflyer X8 [155], and many others.

6.1.2 Motivation

With the advances in electromechanical systems and sensing technologies, recently quadrotors have become popular in unmanned systems research. Their small size and manoeuvrability makes it possible to fly indoors and outdoors. Compared to

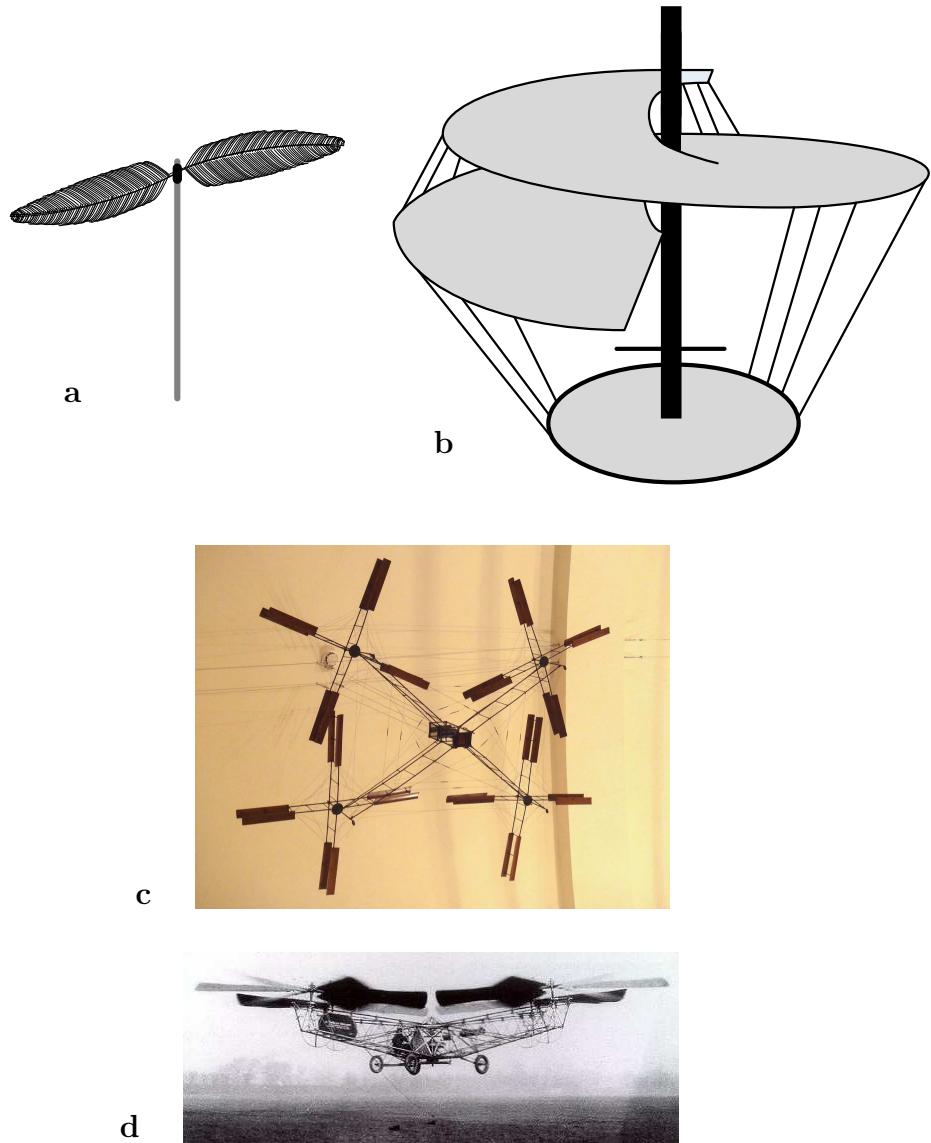


Figure 6.1: The evolution of the quadrotor: **a)** Chinese top, a flying toy made 2500 years ago. **b)** Helix, a design by Leonardo da Vinci (1452-1519) [148]. **c)** Gyroplane No.I, the first quadrotor which could fly 0.6 meters in 1907 [150]. **d)** De Bothezat's quadrotor, built in 1922 in Dayton, Ohio [151].

other rotorcraft, they are less complicated and easier to control. They have many different applications, including border patrol, search and rescue [49], pipeline monitoring, wildfire monitoring, traffic monitoring, land surveys, and agile load transportation [156].

6.1.3 Quadrotor Model

The quadrotor belongs to the family of rotorcraft. A rotorcraft is a flying machine that uses lift generated by rotor blades. Other rotorcraft are different types of helicopters including: the classic helicopter equipped with a main rotor and a tail rotor, the twin rotor or tandem helicopter, and the co-axial rotor helicopter. These helicopters have complex structure and control systems.

Helicopters and quadrotors use thrust generated by rotor blades instead of fixed wings to fly. What makes the quadrotor different from other rotorcraft is the removal of the complex structure and control mechanics such as the swashplate. The quadrotor only has four fixed pitch rotors. Vehicle motion is controlled by altering the rotation speed of rotors instead of the rotors' pitch. Generally, quadrotors use symmetrically pitched blades arranged in pairs. According to Fig. 6.2-a, front and back rotors are rotating clockwise while the left and right rotors are rotating counter-clockwise. By changing the rotation rate of each rotor interdependently, the total thrust and torque of the vehicle can be controlled.

6.1.3.1 Kinematics Model

Fig. 6.3 shows a simplified sketch of the quadrotor, its coordinate systems, and the free body diagram. The frame tagged with $X - Y - Z$ shows the body frame. The frame tagged with $N - E - D$ shows the inertial frame, based on the North-East-Down standard. Assume ϕ , θ , and ψ are Euler angles in the body frame which represent roll (rotation around X), pitch (rotation around Y), and yaw (rotation around Z) angles respectively. These angles are represented by Ω

$$\Omega = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T. \quad (6.1)$$

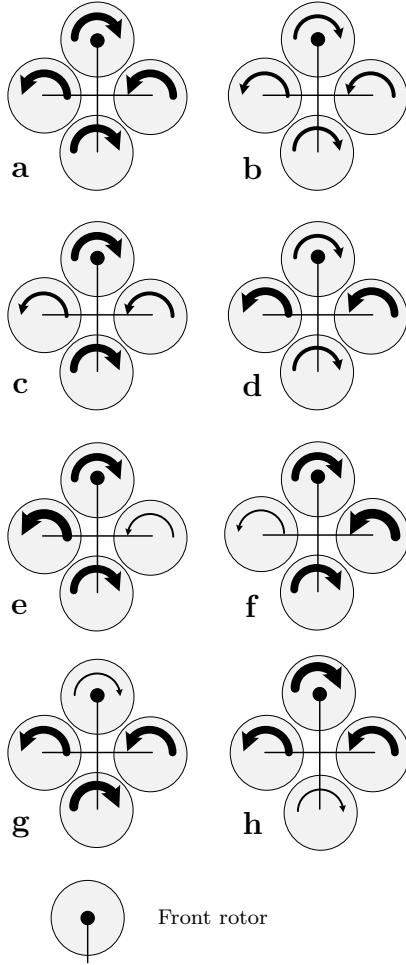


Figure 6.2: By controlling rotor speeds of a quadrotor, it can be flown in different directions. The thickness of each arrow inside the rotation discs correspond to the rotation rate; **a)** vertical take-off, **b)** vertical landing, **c)** yawing clockwise, **d)** yawing counter-clockwise, **e)** rolling right, **f)** rolling left, **g)** pitching forward, and **h)** pitching backward.

To transform a vector from the body frame to the inertial frame, the following transformation is used [55]

$$R = \begin{bmatrix} c\psi c\theta & c\psi s\phi s\theta - c\phi s\psi & s\phi s\psi + c\phi c\psi s\theta \\ c\theta s\psi & c\phi c\psi + s\phi s\psi s\theta & c\phi s\psi s\theta - c\psi s\phi \\ -s\theta & c\theta s\phi & c\phi c\theta \end{bmatrix}, \quad (6.2)$$

where the order of rotation used is yaw, followed by pitch, followed by roll. $c\phi$ and $s\phi$ denote $\cos(\phi)$ and $\sin(\phi)$ respectively, similarly for ϕ and θ .

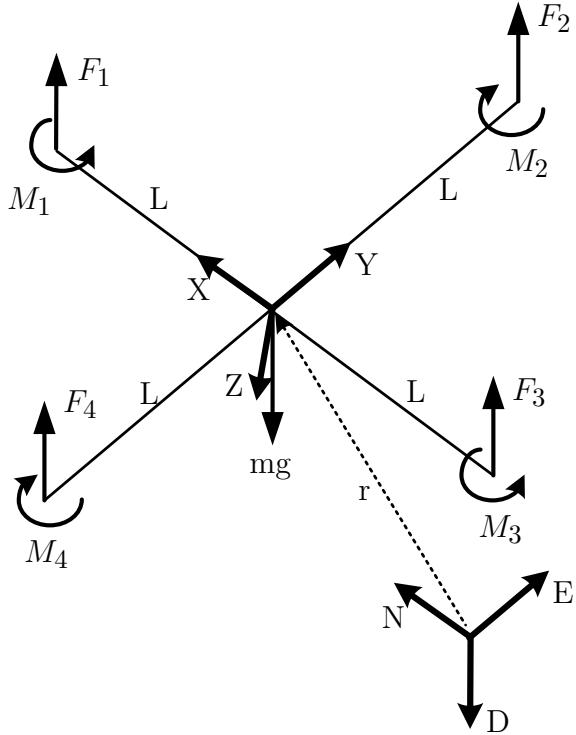


Figure 6.3: The coordinate systems and the free body diagram of the quadrotor with acting forces. The frame shown by $X - Y - Z$ is the body frame. The frame shown by $N - E - D$ is the inertial frame, following the North-East-Down standard. F_i and M_i ($i=1..4$) are the aerodynamic forces and moments produced by the i^{th} rotor, respectively. L is the moment arm, the distance from the center of the quadrotor to the axis of rotation of the rotors. g is the acceleration due to gravity and m is the mass of the quadrotor [55].

Assume ω shows the angular body rates. Then Euler rates, $\dot{\Omega}$, and angular body rates, ω , are related through the following equation [55]

$$\omega = R_r \dot{\Omega}, \quad (6.3)$$

where

$$R_r = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix}. \quad (6.4)$$

6.1.3.2 Dynamics Model

A dynamics model consists of translational and rotational motions. Using the Newton-Euler method and ignoring the gyroscopic moment due to the rotors inertia, the equations of the rotational motion are derived in the body frame as following [53]

$$I\dot{\omega} = -\omega \times I\omega + M_b, \quad (6.5)$$

where I is the moment of inertia matrix of the quadrotor. This matrix is acquired through system identification [157]. M_b is the moments acting on the quadrotor in the body frame and is defined as [53]

$$M_b = \begin{bmatrix} L(-F_2 + F_4) \\ L(F_1 - F_3) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}. \quad (6.6)$$

In equation (6.6), F_i and M_i ($i=1..4$) are the aerodynamic forces and moments produced by the i^{th} rotor, respectively. L is the distance from the center of the quadrotor to the axis of rotation of the rotors. F_i and M_i are both directly proportional to the square of the i^{th} rotors speed.

To derive translational equations, Newton's second law is used. According to Fig. 6.3, the forces on the system are gravity in the D , the Down, direction and the forces from each of the rotors, F_i ($i=1..4$), in the $-Z$ direction. If the position vector of the center of mass of the quadrotor in the inertial frame is shown by r ,

$$r = \begin{bmatrix} x & y & z \end{bmatrix}^T, \quad (6.7)$$

then, following equation governs the translational motion [53, 55]

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ -(F_1 + F_2 + F_3 + F_4) \end{bmatrix}, \quad (6.8)$$

where g is the acceleration due to gravity and m is the mass of the quadrotor.

6.1.4 Advantage and Disadvantage

Small-sized quadrotors have significant advantages specially as an unmanned system. Compared with ground robots, they can move in any direction and therefore are not limited by obstacles. Compared to other rotorcraft such as helicopters, quadrotors have less complex dynamics.

However, for small-size autonomous flying robots, there exists a number of challenges. These challenges impose limitations on flight time, autonomy, and manoeuvrability of the rotorcraft and include the following:

- **Nonlinearity:** Quadrotor models have nonlinear kinematics and dynamics. These models are used for control, stabilization, and state estimation purposes. Most techniques of control or state estimation require linearization of the system model. Linearization is an approximation and can affect the performance of the controller.
- **Fast Dynamics:** The small time-constant of the quadrotor, which is because of fast dynamics, requires a robust and reliable controller.
- **Limited Payload:** Due to limited payload, limited number of sensors can be carried onboard. Moreover, lightweight batteries must be used which limits the flight time.

- **Vibration:** Vibration caused by high speed rotors affects the onboard sensor measurements. To minimize the vibration effects, efficient noise removal filtering and vibration damping must be used.

6.2 Quadrotor Navigation

The main task for an autonomous flying robot is reaching a desired location without human supervision and intervention. In the literature and the robotics community, this important task is referred to as *navigation* or *guidance*. In this section, quadrotor navigation is investigated.

To address the task of navigation, a set of problems must be addressed. These problems include the following:

- Mission planning,
- Map learning: simultaneous planning, localization, and mapping,
- State estimation, and
- Control.

Fig. 6.4 depicts these tasks and their dependency on each other. Each task in each level operates based on the information received from the next higher level. Mission planning determines the general behaviour of the robot. For example, it may decide that the robot will explore an unknown environment, then it will return to the start point and will deliver an exploration report such as a map. Map learning is the task of modelling the world, which requires mapping, localization, and planning a path between waypoints. State estimation is the process of fusing data from all available sources. For instance, using an optical sensor, a robot can estimate 2D pose and heading through localization and mapping. The 2D pose and the heading can be fused by odometry or inertial measurements to provide better estimates. The

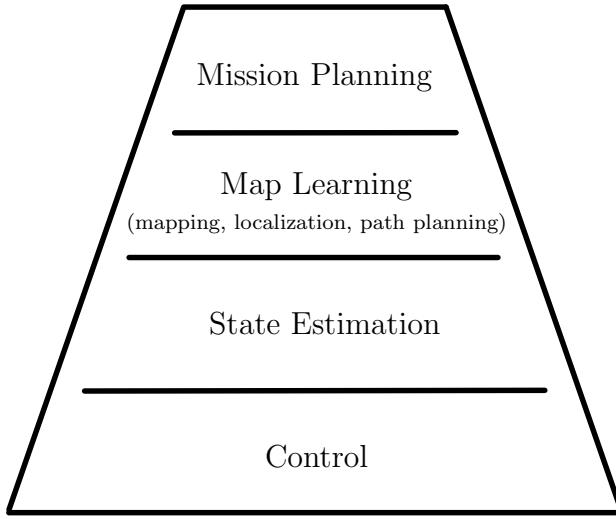


Figure 6.4: Required tasks to accomplish navigation. Each level relies on information received from the next higher level.

estimated state is used by the Control level to generate proper control signals to stabilize the robot towards a desired state. In the rest of this section, each of these problems in the context of the quadrotor is briefly introduced.

6.2.1 Control

Compared to other types of rotorcraft, quadrotors are mechanically simpler and easier to control. However, controlling a quadrotor is still a challenging problem due to its system nonlinearities, cross couplings of the gyroscopic moments and underactuation [55, 158]. For more information on controlling a quadrotor, see [53, 159].

6.2.2 State Estimation

Any control algorithm needs to have access to the real state of the system. The state of a system is a set of variables which describes enough about the system so that one can determine and analyze its future behaviour. In practical applications, state variables are affected by noise and might not be observable directly. To solve these problems, state estimation techniques are used. Accurate state estimation directly

affects the performance of the controller and therefore the overall performance of the system. Typically, the state of a quadrotor includes its orientation, position, and velocities.

6.2.2.1 History

State estimation has a long history; however, its application to quadrotors and other small flying robots is relatively new. Smoothing and filtering are two general approaches for state estimation. Each has its own advantages and disadvantages introduced in Part II.

In 2006, Thrun *et al.* [37] were among the first researchers who did real-time state estimation. They performed state estimation with a helicopter in an outdoor environment. In 2009, Grzonka *et al.* [38] from the University of Freiburg and Bachrach *et al.* [40] from the Massachusetts Institute of Technology (MIT) performed state estimation for a quadrotor, independently. MIT's work won the 2009 International Aerial Robotics Competition (IARC) held by the Association for Unmanned Vehicle Systems International (AUVSI). Since then, researchers have investigated different configurations for state estimation, but one thing which is common in all solutions is the use of Kalman filtering. Kalman filtering, if tuned properly, can provide accurate results.

6.2.2.2 Motivation

Knowledge of the state of a system is necessary to solve many control issues; for instance, stabilizing the heading of a quadrotor to a desired angle requires the knowledge of the true heading angle such that the stabilizer can provide a proper control signal to reduce the error between the desired and the true headings. Usually if the state of the system is observable directly, it requires some signal processing. This extra processing can introduce processing errors which may cause inaccurate estimates.

These estimates need to be filtered or fused by other signals to have better estimates. Nevertheless, in most practical robotic applications, all state variables of the system cannot be observed directly. Instead, indirect effects of other state variables are observed. For example, in indoor environments, linear velocity of a quadrotor is not observable directly; however, one can differentiate the position estimates of the quadrotor to have an estimate of the velocity.

6.2.2.3 State Variables

Depending on what sensors are used on a flying robot, the configuration of state variables might change. For example, if an IMU is used onboard, biases related to the IMU might be a part of the state vector. Nonetheless, the state of a quadrotor needs to have at least pose and orientation information. A complete state would include velocity components, as well. A typical state, which is used by most researchers, is defined as

$$x = \begin{bmatrix} \Omega^T & \dot{\Omega}^T & r^T & \dot{r}^T \end{bmatrix}^T \quad (6.9)$$

where Ω is the Euler angles, r is the position of the robot, and $\dot{\Omega}$ and \dot{r} are the rates, each defined as:

$$\Omega = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T, \quad (6.10)$$

$$\dot{\Omega} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T, \quad (6.11)$$

$$r = \begin{bmatrix} x & y & z \end{bmatrix}^T, \quad (6.12)$$

$$\dot{r} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T. \quad (6.13)$$

ϕ , θ , and ψ are the roll, pitch and yaw Euler angles; $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$ are rates of Euler angles, x , y , and z are Cartesian coordinates of the robot; and \dot{x} , \dot{y} , \dot{z} are velocities of the robot along X , Y , and Z axes respectively. Position and velocity variables are

expressed in the navigation frame.

State variables are usually smoothed or filtered through Bayesian estimation. In the filtering process, mathematical models of the quadrotor are taken into account. Depending on the available resources and the required accuracy, different implementation of Bayesian filtering is used, including extended Kalman filtering, unscented Kalman filtering, and particle filtering. These approaches are thoroughly presented in Part II.

6.2.2.4 Advantage and Disadvantage

State estimation algorithms such as different versions of Kalman filtering are usually fast and real-time, especially when the size of the state vector is small and does not grow over time. The recursive formulation of Bayesian estimation algorithms makes it possible to ignore relatively past estimates and hence, optimize memory usage.

In spite of providing these benefits, state estimation is not generally an easy task. It requires design, tuning of parameters, and processing and memory resources. One of the parameters used in most filtering algorithms is the statistics of noise signals such as mean and deviation of observation or process noises. Finding these parameters requires analyzing sensor measurements in advance. If these parameters are not accurate enough or tuned properly, the state estimation might diverge.

For highly nonlinear systems, linearized filtering algorithms such as the extended Kalman filter may crash due to linearization errors.

Some sensors which are used for state estimation, such as inertial sensors, drift over time. This is a problematic issue and the filtering must be reset periodically. In the rest of the section, pros and cons of different state estimation techniques in relation to the literature, which was already reviewed, are presented.

In 3D helicopter mapping [37], laser odometry, IMU, and GPS measurements are fused using a real-time probabilistic optimization. In the optimization process, the system

model is linearized and noise characteristics are assumed to be Gaussian. These are valid assumptions and make it possible to perform the optimization real-time; however, the helicopter model is extremely nonlinear and the Gaussian approximation can result in inconsistent maps for flights with longer durations or complicated manoeuvres.

In the Mikrokopter project [38], MIT’s work [40], the CityFlyer project [46], the Hector package [48], and the USAR platform [49], the extended Kalman filter is used to fuse data from multiple sources. Kalman filtering has its own pros and cons as mentioned above: it requires minimal memory and is relatively fast but needs tuning parameters and linearization of system equations.

6.2.3 Map Learning

Deploying an autonomous robot in a real world scenario requires learning maps. Learning maps is different than just mapping. In mapping it is assumed that the pose of the mapper is known, but in learning maps, generally, pose is not known. To learn maps, a number of key problems should be addressed, including

- Mapping,
- Localization, and
- Path planning.

Fig. 6.5 depicts these key issues and their relations. Mapping is the process of modelling a robot’s world given sensory measurements, while localization calculates the position of the robot within the map. To move between two given points in the world, a path planning or motion planning algorithm is required.

There is a tight dependency between these components. In an unknown environment, mapping and localization cannot be decoupled. This is because to make a map, the robot needs to know its current position and to know its current position, it needs

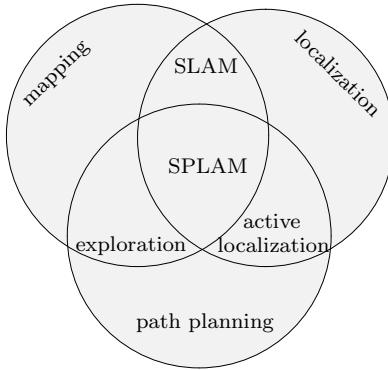


Figure 6.5: Tasks that need to be accomplished towards map learning [160]. An autonomous robot needs mapping, localization, and path planning to complete its tasks efficiently.

to have a map. Simultaneous localization and mapping addresses this issue. In cases where the map is known in advance, active localization algorithms are used to improve the pose of the robot by selecting control actions that will reduce the uncertainty of the robot's pose estimate. If the pose of the robot is known, exploration algorithms are used to find waypoints which lead the robot to the unexplored parts of the environment. The stack of mapping, localization, and path planning is often referred to as simultaneous planning, localization, and mapping (SPLAM); integrated autonomy solutions; or autonomy packages [160]. Complete information on different methods for components of map learning in relation to mapping and localization can be found in [3], and for path planning in [131].

6.2.3.1 History

History of SLAM, in general, is explained in Part II completely. In this section SLAM and path planning are explored in direct relation to quadrotor rotorcraft. SLAM and path planning for indoor quadrotors were first performed in 2009 by Grzonka *et al.* [38] from the University of Freiburg and Bachrach *et al.* [40] from MIT, independently. Later, other researchers continued the same trend to improve on previous results. In

2010, Dryanovski *et al.* [46] at the City College of New York presented their approach for quadrotor SLAM. It was similar to the work proposed by Grzonka *et al.* [38]. In 2011, Kohlbrecher *et al.* [48] from the Darmstadt University of Technology proposed their fast approach for quadrotor SLAM. The latter two solutions had no path planning involved. All these solutions are using scanning laser rangers as a key perception sensor to perform SLAM. Other researchers at the University of Pennsylvania and the Swiss Federal Institute of Technology in Zurich (ETH Zurich) published similar results for SLAM and path planning with the quadrotor.

6.2.3.2 Motivation

Map learning, also called world modelling, which is the knowledge of the environment that a robot exists in, is needed for state estimation. While dynamics and kinematics models govern the motion of the robot, world models govern the level of the autonomy that a robot possesses. Robot models usually do not change by time and place; however, world models are unique in each different environment. This implies that robot models can be learnt in advance, but world modelling has to be performed in real-time. To learn the world model in real-time, a key requirement is the knowledge of the position of the robot. This poses different problems in indoor and outdoor environments.

A global positioning system is a system of satellites which provide accurate positioning information. GPS systems have been operational since 1994. The main challenge with GPS is that it doesn't operate in environments which lack signal reception. Such environments include underwater, indoors, mine tunnels, and extraterrestrial spaces which are referred to as GPS-denied environments. In GPS-denied environments map learning and state estimation becomes crucially important because there is no reference positioning system.

Vicon, a system of multiple fixed cameras, is widely used as an indoor positioning

and localization method. Vicon provides very accurate pose estimates; however, it is an expensive system that requires pre-installation which is a limiting factor.

SLAM, as a perception algorithm, is an alternative solution which relies on laser, vision, sound navigation and ranging, and inertial sensors fusion. It provides relatively accurate estimates; however, it is a complicated problem and requires extra processing capacity and memory.

6.2.3.3 Path Planning

A formal definition of simultaneous localization and mapping is defined in Part II. To define the path planning formally, a few key terms are introduced first.

- **Workspace:** Workspace is the world that contains the robot. It also contains obstacles and free spaces.
- **Free configuration space:** The set of all configurations in the workspace that the robot can achieve and does not involve any contact with obstacles is free configuration space.
- **Path:** Path is a parameterized curve through the free configuration space.

Given these definitions, the task of path planning from a start point, $q = q_s$, to a goal point, $q = q_g$, is defined as finding a path in the free configuration space that connects the start point to the goal point.

Path planning is employed whenever the robot needs to move from one point to another. Path planning is used for different purposes such as: coverage, mapping, localization, and navigation. In this work path planning is used for mapping and navigation.

6.2.3.4 Advantage and Disadvantage

Map learning brings more autonomous capabilities to robots. As an example, imagine a quadrotor navigating outdoors using GPS signals. As a part of the quadrotor's

mission, the robot will enter a building where there is no GPS reception. Without positioning information, it will definitely be lost indoors. If the robot is equipped with map learning techniques including mapping, localization, and path planning, then it can perceive its environment without GPS signals. Not only it will be able to find its way, but also it can perform complicated missions indoors. For example it can look for special targets, pick up important objects, and prepare a floor plan in support of other team members.

Although extremely valuable, map learning techniques are one of the important bottlenecks in robotic applications. With advances in industrial robotics, the reason that robots are less employed in households, restaurants, schools, and hospitals is their limited capabilities in perception and map learning. Often, map learning techniques and specifically the SLAM problem demands high memory requirements and computational power. This becomes more challenging for flying robots, where there are payload limitations, which means lightweight and thus less capable sensors and processing systems should be carried onboard. With all these problems, the map learning algorithm for the quadrotor should be designed in such a way that it can work with limited sensing and processing capabilities, and yet, be able to perform SLAM and path planning with high quality and in 3D.

In the next few paragraphs, pros and cons of different map learning techniques in relation to the literature, which was already reviewed, are presented briefly.

In 3D helicopter mapping [37], there is no localization, path planning, and autonomy involved. It is only mapping and the developed maps visually look consistent. Moreover, the developed maps are composed of point clouds with no uncertainty associated with them. This makes map development fast but it ignores the probabilistic nature of mapping which can be useful for autonomy and path planning.

In the Mikrokopter project [38], GraphSLAM is used for localization and mapping. GraphSLAM is slow since it optimizes the whole trajectory; therefore, the quadrotor

cannot fly fast or fly a long distance. However the advantage of the GraphSLAM is that it enables an efficient loop closure. The path planning is based on D^* , a variation of the A^* algorithm. It has the advantage that it can reuse previous paths to modify an invalid path; however, in general, it is a slower algorithm compared with algorithms like potential fields.

In the CityFlyer project [46], only mapping and localization were performed and there was no path planning or autonomy. The experiment was performed in a small environment with a short flight time. The mapping and localization are based on particle filtering which is a suitable choice for such a nonlinear system; however, this means that no processing can be done onboard.

The work by Bachrach *et al.* [40] also uses particle filtering for SLAM. They performed path planning in the information space; however, their work lacks autonomy at the mission level.

The Hector package [48] also lacks path planning and mission planning; however, their SLAM algorithm is fast, consistent, and accurate, as shown by extensive experiments. In the USAR platform [49], a mission control module is represented in the autonomy solution which controls the actions of the quadrotor. As mentioned above, it uses an extended Kalman filter to fuse visual and laser odometries. They do not generate any geometric map for localization; instead drifts of the quadrotor are corrected by recognizing features in the most recent observations. They do not explain how their path planning is performed and whether it is planned in advance or on-the-fly. Moreover, they have presented only one experiment.

6.2.4 Mission Planning

Mission planning for an autonomous flying robot is defined as a set of actions which should be taken at different times. It is part of the navigation stack and like a state machine, it tells the robot what to do. For example, a simple mission for an

autonomous flying robot which is patrolling over borders between two countries would be: fly over the curve of the border at a given altitude with a given velocity. If a moving object is approaching the border, the flying robot will calculate velocity and position of the object and will report them to headquarters.

An artificial intelligence based solution for this task of navigation is to define a set of navigational behaviours. Each behaviour is a process or control law that achieves and/or maintains a goal [161, 162]. As an example, Obstacle avoidance behaviour maintains the goal of preventing collisions and follow-me behaviour achieves the goal following the position of a person. Some behaviours are prerequisites for other behaviours. For instance, if a robot wants to perform follow-me behaviour, knowledge of the current position of the robot is required. Therefore, localization behaviour is required for following a person.

Path planning is another important behaviour which most other behaviours such as follow-me and return-home rely on. For these behaviours, a path is generated between the goal point and the current position of the robot. For more information on behaviours, see [161].

6.3 Summary

In this chapter, background for autonomous navigation by a quad rotorcraft in GPS-denied environments was introduced. First, the quadrotor and its basic structure, capabilities, and weaknesses were discussed. Then, navigation for the quadrotor was investigated. Control, state estimation, map learning and mission planing are primary tasks of the navigation. Each of these tasks was reviewed and discussed. Map learning, which includes mapping, localization, and path planning, were reviewed in more detail.

Chapter 7

Perception, Navigation, and Autonomy

This chapter presents the proposed solution for autonomous perception and navigation by a quadrotor. In Section 7.1, overview of the system is presented. Section 7.2 introduces the proposed method which includes simultaneous localization and mapping, path planning, state estimation, and mission planning. Section 7.3 presents experimental results in simulated and real-world environments. Finally, Section 7.4 summarizes the chapter.

7.1 System Overview

To enable a quadrotor to navigate autonomously, a set of interdependent requirements must be addressed, including control, state estimation, map learning, and mission planning. As mentioned in Chapter 6, autonomous navigation requires stabilization of the vehicle. To stabilize the vehicle, the state of the vehicle must be known. In GPS-denied environments, state estimation is calculated using simultaneous localization and mapping algorithms. Once the state of the vehicle is known, depending on the planned mission, the vehicle can move from one point to another using a path planning

algorithm. This chapter proposes an integrated solution for all these requirements. To perceive the environment, sensors play a key role. For the autonomous quadrotor, a sensor suite including inertial, laser ranging, and optical sensors are used to provide exteroceptive and proprioceptive measurements. All four levels of the autonomy (Fig. 6.4) rely on the information from these sensors, directly or indirectly. The state estimation is designed so that if GPS signals are available, they can be integrated with the solution. The sensor suite with the developed algorithms are independent of the hardware platform. In other words, the proposed autonomy solution with the sensor suite can easily be ported to other robots such as ground robots. The only necessary change would be modifying the control level, in the navigation stack (see Fig. 6.4), to apply the proper control signals to the specific actuators of the robot. As will be shown in the experimental results, portability of the autonomy solution and the sensor suite have been demonstrated in real-world experiments on a ground robot and a quadrotor.

7.2 Proposed Perception and Autonomy Solution

In this section, the problem of developing an autonomous quadrotor is defined formally. Advantages and disadvantages of previous methods are reviewed. Then, the proposed solution is explained in detail. Finally, some discussions followed by contributions of the work are presented.

7.2.1 Problem Statement

Developing an autonomous quadrotor in GPS-denied environments is considered to be a challenging problem. Formally and briefly, this problem can be defined as *to design and develop a quadrotor capable of performing a set of tasks in GPS-denied environments, autonomously*. An autonomous quadrotor needs the following tasks to

navigate: control, state estimation, map learning, and mission planning. Each of these tasks is required to achieve autonomy. More specifically, map learning involves path planning, localization, and mapping. The mission might be designed by a human, but the robot must be able to transition between different stages of the mission. The main challenge is that all these tasks must be performed in spite of the characteristics and limitations of a quadrotor, such as nonlinear dynamics, small time-constant, limited payload, vibration effect, and 3D motion. The sensor suite should enable the robot to have enough sensing from the environment for map learning, while being designed with consideration of the limitation of the quadrotor. Moreover, the developed solution must run in real time.

7.2.2 Problems with Existing Methods and Motivations

Most research that demonstrates outstanding results with quadrotors, such as the work by Nathan *et al.* [53] and Juggling Quadrotors by ETH [163], use Vicon system. Vicon provides very accurate positioning information and avoids dealing with the challenging problem of map learning; however, this is not a problem that can be solved by Vicon in every desired indoor environment: Vicon is an expensive system and needs pre-installation. Therefore, relying on Vicon can provide spatially limited autonomy.

Some studies lack path planning and exploration as an important part of map learning. For instance, in [46, 48], a pilot is flying the quadrotor, and the robot only performs mapping and localization.

Planning a mission is also another important task that most researchers ignore. The quadrotor should be able to interact with the environment based on the priority of tasks and transition from one task to another autonomously.

To enable the quadrotor to operate autonomously in every indoor environment, map learning, as alternative solution for GPS and Vicon, is the main focus of this work.

Moreover, path planning and exploration are integral to the solution. Mission planning and transition between different behaviours and tasks are also important parts of the work. All these elements provide a complete autonomy package for a quadrotor.

7.2.3 Description of the Method

An overview of the proposed autonomy system is shown in Fig. 7.1. The proposed solution is composed of two main modules: *Mission Planner* and *Autonomy*. Each module consists of several blocks. *Mission Planner* takes care of sequencing the autonomous behaviours. The *Autonomy* module accepts behaviours from *Mission Planner* and takes actions to achieve or maintain the goal of the behaviour.

The sensor suite includes an IMU, a scanning laser rangefinder with a horizontal scan, a second scanning laser rangefinder with a vertical scan, a Kinect camera, an altimeter, and a GPS. Sensors connected by a dashed line to the autonomy blocks are optional sensors. In *2D SLAM*, a 2D view-based SLAM is performed using the horizontal scanning laser rangefinder and the IMU. This block outputs 2D Cartesian coordinates and yaw angle. In the *rgbdSLAM* block, a Kinect camera is used to do 3D SLAM. Using the accelerometers of the IMU, in the *KF Fusion* block, the results of these two blocks and the measurement from the altimeter are fused by a Kalman filter. If there exists any GPS measurement, it is fused with the estimated pose to provide an estimation with bounded error. The estimated pose is then used by the *Planner* block to find waypoints and plan paths between waypoints. The calculated waypoints and position feedback are passed to the *controller* block to drive the quadrotor. *Obstacle Avoidance* block has the highest priority and directly uses the laser ranger to avoid obstacles. Although in the *Planner* block, the path planning algorithm makes plans taking into account the obstacles; however, dynamic obstacles or situations caused by disturbances or controller inaccuracies require an obstacle avoidance plan with a higher laser scan rate. In the *voxel mapping* block, a 3D volumetric map is generated

by the vertical scanning laser rangefinder using the filtered pose.

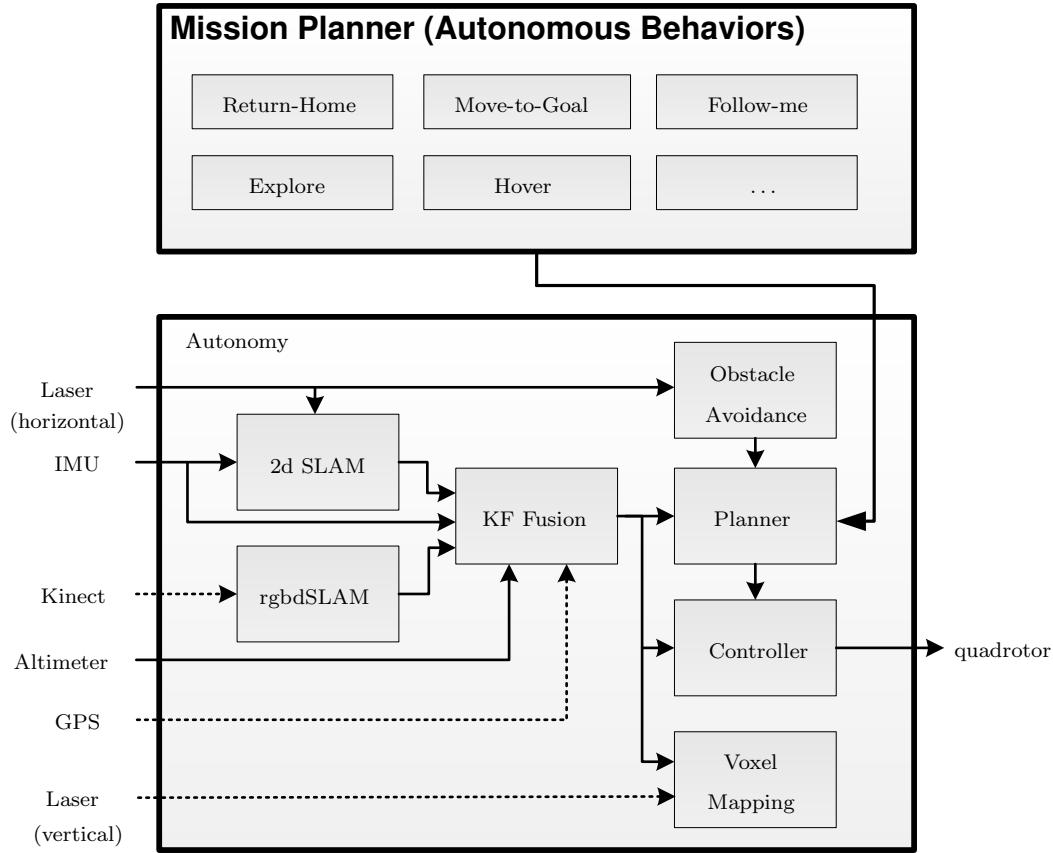


Figure 7.1: Proposed autonomous navigation in GPS-denied environments is composed of two main modules: *Mission Planner* and *Autonomy*. *Mission Planner* takes care of organizing and sequencing the autonomous behaviours. The *Autonomy* block accepts these behaviours and takes proper actions. In the *Autonomy* block, an IMU, a horizontally mounted laser ranger, and an altimeter are minimum required sensors. These sensors are connected by solid lines to the related blocks. A vertically mounted laser ranger, a Kinect camera, and a GPS are optional sensors, connected by dotted lines to the related blocks. In the *2D SLAM* and *rgbdSLAM* blocks, view-based SLAM and feature-based SLAM are performed respectively. In the *KF Fusion* block, the results from the SLAM blocks are fused with the IMU, GPS, and the altimeter measurements to generate a pose estimate. The estimated pose is used by the *Planner* block to generate a waypoint. The waypoint is used by the *Controller* to fly the rotorcraft. The *Voxel Mapping* block, makes a 3D map using the estimated pose and the vertical laser ranger. The horizontal laser ranger is also used to avoid obstacles.

7.2.3.1 Autonomous Behaviours

Autonomous behaviours are a set of behaviours designed to navigate the robot efficiently. These behaviours rely on exploration, path planning, and obstacle avoidance behaviours. Behaviours such as *follow-me*, *return-home*, *move-to-goal* and *return-to-me* are based on path planning between two given points and following the path. *Obstacle avoidance* behaviour is performed at two levels. First, it is performed at the map level, where occupied cells are dilated and an optimal path is designed. Second, it is performed using the laser ranger and keeping a safe distance from instantaneous detected obstacles which are closer than a pre-specified threshold.

Any complicated mission can be represented as a set of behaviours. A sample mission composed of the mentioned behaviours is presented in Fig. 7.2. In this mission (mapping, localization, and path planning are not shown for clarity), first the robot explores the environment (*explore* behaviour). Once exploration is complete, it moves to a given goal point (*move-to-goal* behaviour). Once it reaches the destination, it returns to the start point where the mission was started (*return-home* behaviour).

Fig. 7.2 depicts the state machine of the navigation. Two behaviours, obstacle avoidance and hold (as an emergency stop), are running at a higher priority than others.

7.2.3.2 2D SLAM

The quadrotor can fly at a fixed altitude; therefore, a 2D grid map is generated and used for navigation. The horizontal scanning laser ranger is used to perform 2D SLAM and generate an occupancy grid map. The method used for 2D SLAM is adopted from [48] which is accurate and fast. In this method the scans are transformed into the stabilized local frame given the roll and pitch angles. These angles are estimated through an attitude and heading reference system (AHRS). Utilizing bilinear filtering, scans are matched against the current map at the rate of 40 Hz.

The mapping process seeks to calculate a transformation between a current scan and

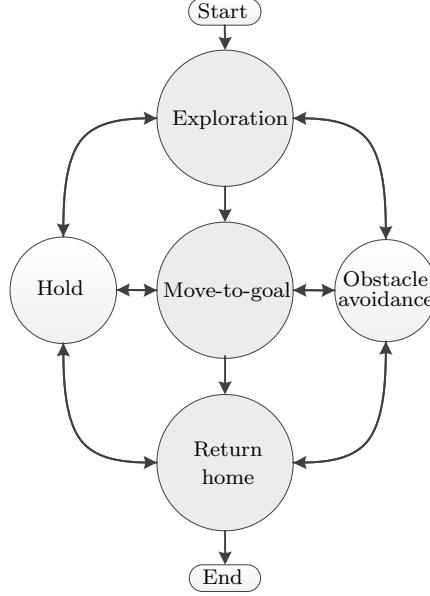


Figure 7.2: A sample mission composed of basic navigation behaviours. First, the robot explores an unknown environment, then it moves to a given goal. Once it arrives at the goal, it returns home. While performing these behaviours, obstacle avoidance and hold (as an emergency stop) are running at a higher priority than others.

the map. Assume the transformation is represented by $\delta = (x, y, \psi)^T$. For the i^{th} beam of the scan, if the transformed end point of the scan is represented by $s_i = S_i(\delta)$, then the map value at the end point is shown by $m(S_i(\delta))$, where $m(\cdot)$ represents the occupancy values from the occupancy grid map. Obviously, s_i should be an occupied cell, since it is the end point of a laser beam. Therefore, the map value at s_i should be compared with an occupied cell. This results in minimizing $(1 - m(S_i(\delta)))^2$, where 1 represents a fully occupied cell. By applying this to all beams, the transformation should minimize the difference between the current map and the new transformed scan as follows:

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \sum_{i=1}^n (1 - m(S_i(\delta)))^2, \quad (7.1)$$

where n is the number of scan beams and δ^* is the desired outcome, which is the required transformation to fit the scan into the map. This problem is formulated and solved in [48] using the gradient ascent approach. To do this, first order Taylor

expansion is applied to equation (7.1) and the resulting Gauss-Newton equation is solved.

To speed up the mapping process and avoid local minima, a multi-resolution map representation is used [48]. The multi-resolution map representation can also be used for path planning and navigation purposes.

7.2.3.3 3D GraphSLAM

Features of the environment can be used to calculate the pose of the robot. To perform feature-based SLAM, at least one camera is needed. The result from the feature-based SLAM can act as an extra source of information for data fusion by Kalman filtering. In this work, feature-based SLAM is accomplished by a Kinect camera using the rgbdSLAM algorithm.

An RGB-D camera is a sensing system that captures RGB images with depth information for each pixel. Generally, RGB-D cameras use either stereo technology [164] or infrared (IR) time-of-flight sensing [165]. Kinect [166] is an IR-based RGB-D camera developed by PrimeSense which has recently attracted the attention of researchers. rgbdSLAM which uses an RGB-D camera, is a robust feature-based SLAM algorithm in which features of the environment are extracted and used for localization and mapping. Inputs of the rgbdSLAM are colour (RGB) and depth images, captured simultaneously. These two images are processed to extract the pose of the camera and build up the map of the environment. Fig. 7.3 shows the required processing blocks of the rgbdSLAM [103, 104]. The first three blocks, speeded up robust feature (SURF), random sample consensus (RANSAC), and generalized iterative closest point (GICP), provide accurate visual odometry. The last block, HOGMAN, provides global consistency of transformations and detects loop closures. In the visual odometry, first SURF features are extracted from the RGB image. Based on the correspondence of the depth values of the features, the transformation between images is extracted using

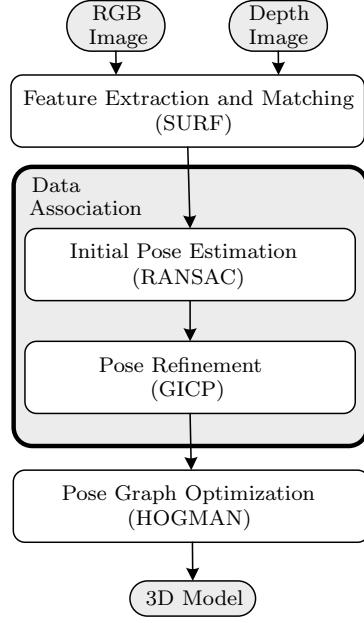


Figure 7.3: Flow chart of rgbdSLAM [104]. Data association in rgbdSLAM is composed of two components, RANSAC and GICP. RANSAC provides an initial estimate of the transformation matrix. The initial estimate is used in the batch data association of GICP to refine the results.

RANSAC. This is an initial estimate and needs to be refined by the GICP algorithm. Here each block is explained in detail.

7.2.3.3.1 Feature Extraction and Matching. To extract features from images, the SURF method is used. This method is based on scale-invariant feature transform (SIFT), but it is several times faster and more robust against different image transformations. SURF relies on sums of approximated two dimensional Haar wavelet responses and integral images. The extracted features are matched against features from the previous image. Then using the depth image at locations of the corresponding features, a set of point-wise and 3D correspondences between frames is generated. This set is used for localization by further processing.

7.2.3.3.2 Initial Estimation. The 3D matching features are used to find the relative transformation between the frames using the RANSAC method. RANSAC

estimates a mathematical model from a set of observed data iteratively. The data is assumed to be noisy and contain inliers and outliers. This can produce false models in case the assumption does not hold. The extracted transformation through this method is not accurate and needs to be refined. The next block uses this estimate as an initial estimate to refine the results.

7.2.3.3.3 Estimation Refinement. Data association is the key element of localization and mapping. This becomes more important when SLAM relies on features only. Correct localization relies on finding correct correspondences between observations and the available map. Incorrect associations may cause pose estimates to rapidly diverge.

A significant improvement in robustness and accuracy of data association is achieved by using batch data association. In batch data association a set of observations is assigned at once, instead of individual assignment and association. By this approach geometric relations of features are taken into account and association distinction will be based on their combined association likelihoods.

If the observed features are internally correlated (which is the case in most SLAM applications), then batch data association techniques can improve the results. As explained in rgbdSLAM and shown in Fig. 7.3, GICP [105] is the technique used for batch data association. GICP is initialized with RANSAC which is required to provide accurate results. GICP takes two sets of features as inputs with an initial estimate of their relative transformation. Unlike most other techniques, GICP takes into account the internal geometry of the features by assigning a plane to each set and produces more accurate results. GICP is based on the locally planar structure of point clouds where plane-to-plane distance is used instead of point-to-point distance. In this method, maximum likelihood estimation is used to estimate the required transformation iteratively.

The generalized iterative closest point algorithm is presented in Algorithm 7.2.1. The operator \oplus is the transformation operator, and d_{thr} is a threshold to remove outliers. Given two sets of features, F_1 with N features and F_2 with M features, the result is a transformation presented in line 13. The transformation is resolved using maximum likelihood estimation [105]. In line 1, the transformation is initialized with the value obtained from the RANSAC algorithm. In lines 2 and 3, A and B , which will hold corresponding features from F_1 and F_2 , are initialized with empty sets. Then the algorithm runs until it converges or the number of iterations exceeds a given value. In line 6, for each feature in F_1 , the closest feature from F_2 is found. If the distance between these two features is less than a threshold (line 7), then both features are added to sets A and B , respectively. Then in line 10, d_i , the error of the transformed point for each two corresponding members of A and B , is calculated. After performing this loop for all features, in line 13, the transformation is calculated. The superscript $'$ on a vector or a matrix denotes its transpose. C_i^A and C_i^B , $i = 1,..N$, are covariances of members of sets A and B , respectively.

7.2.3.3.4 Pose Graph Optimization. The global and consistent mapping of the SLAM algorithm is based on HOGMAN [94]. HOGMAN is a 2D/3D optimization approach for GraphSLAM. It considers the underlying space as a manifold, not an Euclidean space. The method is accurate and efficient, and it works well in online operations.

In GraphSLAM, as a solution for full SLAM, poses of the robot are represented as nodes in a graph. The edges between nodes are modelled with motion and observation constraints. These constraints need to be optimized to calculate the spatial distribution of the nodes. As mentioned, in this work, this optimization is performed using HOGMAN. More information about HOGMAN can be found in Chapter 3.

Algorithm 7.2.1 Generalized Iterative Closest Point (GICP) algorithm for two sets of features, F_1 and F_2 .

Input: $F_1 = \{f_{1i}\}_{i=1}^N$ and $F_2 = \{f_{2i}\}_{i=1}^M$: Two sets of features,

T_0 : an initial transformation,

d_{thr} : distance threshold.

Output: T : Transformation between F_1 and F_2 .

```

1:  $T \leftarrow T_0$ 
2:  $A = \{a_i\}_{i=1}^n \leftarrow \emptyset$ 
3:  $B = \{b_i\}_{i=1}^m \leftarrow \emptyset$ 
4: while not converged or for a number of iterations do
5:   for  $i = 1 \rightarrow N$  do
6:      $m_i \leftarrow$  closest point to  $T \oplus f_{2i}$  from  $F_1$ ,
7:     if  $\| m_i - T \oplus f_{2i} \| \leq d_{thr}$  then
8:        $A \leftarrow A \cup m_i$ 
9:        $B \leftarrow B \cup f_{2i}$ 
10:       $d_i \leftarrow a_i - T \oplus b_i$ 
11:    end if
12:   end for
13:    $T \leftarrow \underset{T}{\operatorname{argmax}} \sum_{i=1}^N d_i' (C_i^B + T C_i^A T')^{(-1)} d_i$ 
14: end while
```

7.2.3.4 State Estimation

Results from SLAM are fused with the information from IMU by a Kalman filter to provide state estimation. This estimate is used by the planner and controller blocks. Prior to using IMU measurements in the Kalman filtering, they are filtered by an AHRS system; therefore, IMU measurements are not present in the state vector of the system. The state of the system is defined as

$$x = \begin{bmatrix} r^T & v^T \end{bmatrix}^T, \quad (7.2)$$

where r is the position of the robot and v is the velocity of the robot.

$$r = \begin{bmatrix} x & y & z \end{bmatrix}^T, \quad (7.3)$$

$$v = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T. \quad (7.4)$$

State prediction is performed using the accelerometers of the IMU, $a = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$, given the following system equations

$$\dot{r} = v \quad (7.5)$$

$$\dot{v} = Ra + g, \quad (7.6)$$

where g is the constant gravity vector and R is the direction cosine matrix, introduced in Chapter 6, equation (6.2). Euler angles from the AHRS are used in the direction cosine matrix. The GPS, altimeter, visual and laser updates are used for updating the state prediction. Details of Kalman filter are explained in Part II.

7.2.3.5 Planner: Exploration of Unknown Space

To explore an unknown environment, the frontier algorithm is used. This algorithm uses unknown cells located in the boundaries of the known cells as *frontiers* and moves towards them [44]. Algorithm 7.2.2 explains the frontier exploration. In line 1, set U which includes all frontier cells and set C which holds clusters of frontier cells are initialized to be empty. According to the algorithm, first frontier cells are extracted (line 2). Then the frontier cells are clustered based on the connectivity-8 neighborhood (line 3), where the connectivity-8 neighborhood means that all eight cells around a cell which touch either a corner or an edge of the cell, are considered as neighbors of the cell. In line 4, m gets the cardinality of set C (indicated by $|C|$). For each cluster, first the center of the cluster is calculated (line 6); then the distance between the center of the cluster and the robot is calculated (line 7). (x_c^i, y_c^i) is the center of the i^{th} cluster, $i = 1..m$, and d_c^i is the distance of the cluster from the robot. The number of cells in each cluster is calculated in line 8, shown by n_c^i for the i^{th} cluster. Finally, the center of a cluster that has more frontier cells and is closer to the current position of the robot is chosen as the next waypoint. This means that a cluster is chosen which

Algorithm 7.2.2 Frontier exploration.

Input: m : partially explored map,
 (x_r, y_r) : global coordinates of the robot.

Output: c_{wp} : waypoint

```
1:  $U, C \leftarrow \emptyset$ 
2:  $U \leftarrow$  frontier cells.
3:  $C \leftarrow$  clusters of  $U$ .
4:  $m \leftarrow |C|$ 
5: for  $i = 1 \rightarrow m$  do
6:    $(x_c^i, y_c^i) \leftarrow$  centroid of the  $i^{th}$  cluster
7:    $d_c^i = \sqrt{(x_r - x_c^i)^2 + (y_r - y_c^i)^2}$ 
8:    $n_c^i \leftarrow$  number of cells of each cluster
9: end for
10:  $c_{wp} \leftarrow \text{argmax}_{c^i} \frac{n_c^i}{d_c^i}.$ 
```

can minimize the ratio of $\frac{n_c^i}{d_c^i}$. Because of the perception range of the sensors, there is no need to wait for the robot to get to the target waypoint. Therefore, this process can continue and update the waypoints at fixed time intervals or distances. In this work, waypoints are updated every 4 metres. This has the advantage that the robot does not show oscillatory behaviours between waypoints. Once the next waypoint for the exploration is known, a path planning algorithm is used to guide the robot to the new waypoint.

7.2.3.6 Planner: Path Planning

To navigate between two given points, the wavefront algorithm is used. This is performed using the map generated while exploring the environment. The wavefront produces an optimal solution and no local minima are generated [131]. To avoid getting close to the obstacles, occupied cells are dilated such that the planner generates a path with a safe distance from obstacles (line 1). The details are given in Algorithm 7.2.3. The free space is represented by a grid, marked with 0 as unvisited (line 5). The occupied and dilated cells are marked with 1 (line 6). The algorithm generates a wave from the goal cell. The goal cell is marked as a visited cell and

given a value of 2 as the start point of the wavefront (line 7-8). The algorithm then iteratively finds unvisited (marked with 0) neighbors of the wave cells and assigns them values of one greater than the visited wave cell (line 10-12). The result will be a ‘wavefront’ radiating outwards from the goal cell as new cells are assigned increasing values. Once the wave reaches the start point, the planner travels back on the wave using gradient descent and generates the path (line 14-19).

Fig. 7.4 shows an example of the wavefront path planning algorithm. In Fig. 7.4-a, a simulated environment with obstacles is depicted. Obstacles are shown in black and the free space is shown in white. The start point is marked with a green circle and the goal point is marked with a red square. A path should be designed from the start point to the goal point. The dilated areas are shown in gray. In Fig. 7.4-b, obstacles are dilated to avoid getting too close to the obstacles. In Fig. 7.4-c, a wave is generated from the goal point to the start point. The wave is colour coded. Finally, Fig. 7.4-d shows the path designed using the generated wave.

7.2.3.7 Obstacle Avoidance

Obstacle avoidance is achieved using laser beams directly to avoid any unexpected collisions caused by disturbance or dynamic objects. Laser beams are filtered by a sliding window to remove noisy measurements. Then beams are divided into b bins (for example, for the Hokuyo UTM-30LX laser ranger, 54 bins are used, each covering 5°). The range of a bin is defined as the average range of laser beams in that bin. A collision bin is one which identifies an obstacle within a given range. This bin has collision risks. A free bin has a range greater than a threshold. The free bin is the bin used to avoid the collision. If a bin is identified as a collision bin, then the current waypoint is changed such that the new waypoint has 180° offset from the collision bin.

Algorithm 7.2.4 summarizes the obstacle avoidance. Inputs to the algorithm are: the

Algorithm 7.2.3 Wavefront path planning

Input: m : map, q_{start} : start cell, q_{goal} : goal cell, r_{dilate} : dilation size

Output: p : path

```
1: dilate occupied cell for  $r_{dilate}$  cells
2: if  $q_{goal}$  or  $q_{start}$  are located within the dilated cells then
3:   relocate them to the closest free cell.
4: end if
5: mark all free space with 0, as unvisited
6: mark all occupied, dilated and unknown cells with 1
7:  $index \leftarrow 2$ 
8:  $q_{goal} \leftarrow index$ 
9: while  $q_{start}$  not visited do
10:   set all free cells neighboring a cell with value  $index$  to  $index + 1$ 
11:   mark all cells with value  $index + 1$  as visited
12:    $index \leftarrow index + 1$ 
13: end while
14:  $q \leftarrow q_{start}$ 
15: add  $q$  to  $p$ 
16: while  $q_{goal}$  not in  $p$  do
17:    $q \leftarrow$  neighbor of  $q$  with minimum value
18:   add  $q$  to  $p$ 
19: end while
```

scan s , the current waypoint w_c , number of bins b which is used to divide the scan into b sectors, and angular span of the scan α . The output is a new waypoint w_n . If the robot is too close to an obstacle, a waypoint in the collision-free zone of the scan is produced. Otherwise, the waypoint is not changed. In line 1, the collision bin is initialized. In line 2, the free bin is initialized. w_n is calculated from this bin. In line 3, the scan is filtered by a low pass filter to remove noisy spikes. In line 4, the scan is divided into b bins, and each bin is checked to see if there is any risk of collision. This is performed by function $\text{checkCollision}(\cdot)$. If there is a risk of collision with a bin, then the identification number (ID) of the bin is returned for further processing. Based on the ID of the collision bin, a free bin is selected. To do this, first, a candidate bin is chosen with 180° angular offset from the collision bin. If the candidate bin is also a collision bin, the closest collision-free bin to the candidate bin is selected as the free bin. This operation is performed by function $\text{findFreeBin}(\cdot)$

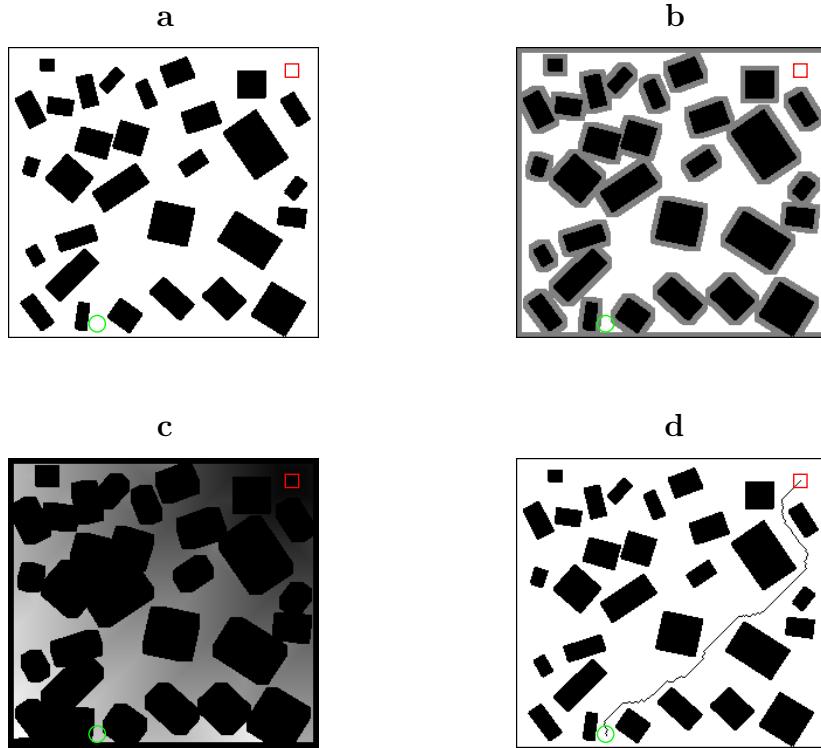


Figure 7.4: An example of the wavefront algorithm. **a)** A simulated environment with obstacles. Obstacles are shown in black while the free space is shown in white. A path should be designed from the the start point, the green circle, to a goal point, the red square. **b)** Obstacles are dilated to avoid getting too close to the obstacles. The diluted areas are shown in gray. **c)** A wave is generated from the goal point to the start point. The wave is colour coded. **d)** Using the generated wave, a path between the start and goal points is designed, shown by a black curve.

in line 6. In line 7, the next waypoint from the collision-free bin is calculated. The bearing of the new waypoint with respect to the current pose of the robot is equal to the orientation of the free bin. The range of the new waypoint with respect to the current position of the robot is a fixed number. In this research, it is set to be 1 m. Once the robot is out of collision risk, it resumes its previous behaviour.

Algorithm 7.2.4 Obstacle avoidance

Input: s : laser scan, w_c : current waypoint, b : number of bins, α : angular coverage of a scan

Output: w_n : next waypoint

```
1:  $collision\_bin\_id \leftarrow \emptyset$ 
2:  $free\_bin\_id \leftarrow \emptyset$ 
3:  $s \leftarrow filter(s)$ 
4:  $collision\_bin\_id \leftarrow checkCollision(s, b)$ 
5: if  $collision\_bin \neq \emptyset$  then
6:    $free\_bin\_id \leftarrow findFreeBin(collision\_bin\_id, \alpha, b)$ 
7:    $w_n \leftarrow waypoint(free\_bin\_id)$ 
8: else
9:    $w_n \leftarrow w_c$ 
10: end if
11: return  $w_n$ 
```

7.2.3.8 3D Voxel Mapping

The 3D voxel mapping is based on a compact and flexible 3D mapping, known as Octomap [167]. Octomap uses octree mapping. An octree is a tree-like data structure where each node can have eight children. In an octree structure, a three dimensional space is subdivided into eight octants recursively. Similar to the 2D occupancy grid mapping, Octomap takes into account the probability of the occupancy of each voxel. This capability makes it a probabilistic map which can be used in dynamic environments. The occupancy probability of a voxel v given the sensor measurements from time 1 to t , $z_{1:t}$ is estimated by [167]

$$p(v|z_{1:t}) = \left(1 + \frac{1 - p(v|z_t)}{p(v|z_t)} \frac{1 - p(v|z_{1:t-1})}{p(v|z_{1:t-1})} \frac{1 - p(v)}{p(v)} \right)^{-1} \quad (7.7)$$

where $p(v)$ is the initial state of the voxel. Octomap is a flexible mapping method which does not need to know the extent of the map in advance. In fact it expands dynamically as required. Since it models occupied, free, and unknown octants, it can be used in exploration and path planning applications. In this work, the vertical laser ranger is used to generate a 3D Octomap.

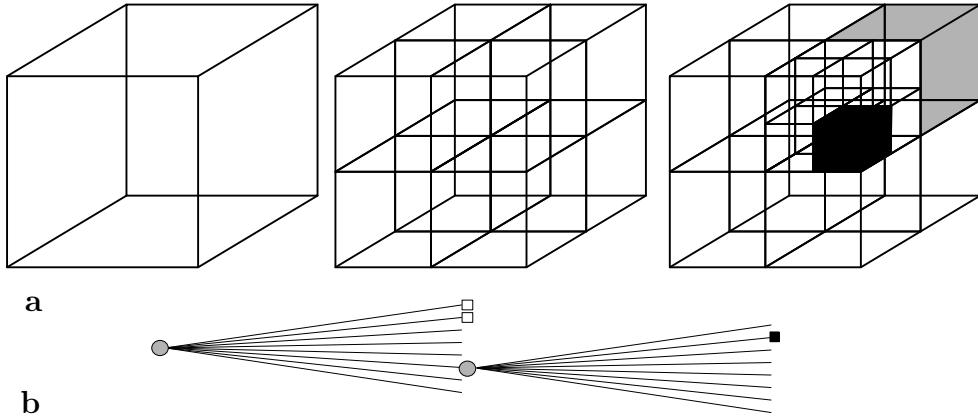


Figure 7.5: An example of an octree structure. **a)** Occupied and free octants depicted in black and gray. Other octants are unknown. **b)** The corresponding tree representation [167].

7.2.3.9 Controller

The controller block accepts waypoints generated by the planner block and adjusts rotor speeds. Nagaty *et al.* [55] developed a cascaded controller for a quadrotor which is used here. The controller is composed of inner loop and outer loop PID controllers. The inner loop controller stabilizes roll, pitch, yaw and altitude while the outer loop controller is responsible for position tracking or waypoint following.

During flight tests, there is the risk of damaging the flying robot. It is very important to have the option of taking over the control of the robot manually, anytime, anywhere.

7.2.4 Advantages and Disadvantages

In this section, advantages and disadvantages of each of the components of the autonomy solution are reviewed separately.

Mission planning is composed of a set of basic behaviours which can be used to create a mission. The advantage of using basic behaviours is obvious: almost any type of mission is composed of a set of basic behaviours. The core of these behaviours is dependent on planning and following a path from one point to another point. This, in turn, relies on the knowledge of the position of the robot within the environment,

which is taken care of by SLAM. For instance, *follow-me* and *return-home* are two different behaviours but in fact they are very similar. The first one, requires a path from the current position of the robot to the position of a person. The second one, needs a path from the current position of the robot to the start point of the mission. The path planning algorithm, wavefront, is a fast algorithm which avoids local minima and generates an optimal path [131]. Moreover, it is relatively easy to implement; however, there are two problems with this algorithm. First, it does not generate a smooth path. Second, it creates paths which are sometimes very close to the obstacles. The first problem is solved during path following. The path following process selects waypoints from the path which automatically smoothes the path. For the second problem, obstacles are dilated to avoid the path coming too close to the obstacles. Also, this algorithm relies on the map of the environment; thus, if a goal point is given to the algorithm which is out of the scope of the map, then it fails to generate a path.

The 2D SLAM algorithm is a fast and robust method. Extensive experiments with a ground robot and the COBRA quadrotor demonstrated its performance. Quadrotors have fast dynamics; therefore, it is important to provide pose estimates as frequently as possible. The 2D SLAM algorithm operates at 40 Hz which is an acceptable rate to control a quadrotor.

rgbdSLAM is a solution based on GraphSLAM. Generally, GraphSLAM solutions tend to be slow. While 2D SLAM operates at 40 Hz, rgbdSLAM performs at about 1Hz. However, rgbdSLAM provides a 3D solution which recovers Cartesian coordinates and the orientation of the robot. Also, rgbdSLAM relies on features; therefore, if the environment lacks features, it may fail.

The Kinect camera is a part of the sensor suite which is used for rgbdSLAM. The main problem with the Kinect camera is its weight, about 400 g, which is a relatively heavy load for a typical quadrotor. There is at least one laser ranger onboard which

weighs about 370 g. To fly a Kinect camera with the laser ranger, a quadrotor with larger payload is needed.

7.2.5 Contribution

The contribution of this work is an integrated autonomy solution. The proposed solution includes all necessary elements to have an autonomous quadrotor. These elements include mission planning, map learning, and state estimation. Furthermore, map learning consists of localization, mapping, and path planning components. The selected sensor suite enables the quadrotor to operate in GPS-denied environment. The proposed solution with the sensor suite can also be used on ground robots or other types of rotorcraft which need autonomy in GPS-denied environments. Details of these contributions are listed as follows:

- Main building blocks of the autonomy, including mission planning, map learning, state estimation, and control, are implemented and integrated to achieve autonomous operations.
- A set of autonomous behaviours such as follow-me, explore, and return-home are developed and structured to enable the quadrotor to achieve autonomous navigation at the mission level.
- Using the mounted sensor suite, perception, which includes localization and mapping, is accomplished in 3D.
- To enable obstacle free flight between waypoints, path planning is realized using the wavefront algorithm.
- By integration of localization, mapping, and path planning, exploration, which is a key requirement for flying robots to navigate in unknown environments, is developed based on the frontier algorithm.

- The proposed autonomy solution and the sensor suite are applicable to other robots such as ground and other small and low-altitude flying robots.

7.3 Experiment

To test the proposed perception and navigation solution, the mission shown in Fig. 7.2 based on autonomous behaviours was implemented. The experiments were implemented and tested in three different configurations, including:

- Simulation in Gazebo,
- Real-world experiment: unmanned ground vehicle, and
- Real-world experiment: quadrotor rotorcraft.

Each experiment is explained in detail.

7.3.1 Data Collection

Prior to performing real-world experiments, data from sensors of a remotely controlled flying quadrotor was recorded. The purpose of data collection is to save time in dealing with hardware and to enable the comparison of different algorithms. The data were collected in standard robot operating system (ROS) [142] format. ROS is a robotic software framework which provides developers with functionalities similar to operating systems. ROS was first initiated by the Stanford Artificial Intelligence Laboratory in 2007. Since 2008, Willow Garage has continued the development of ROS. As of now, ROS is the de facto development platform for most robotic applications.

7.3.1.1 COBRA Quadrotor

The COBRA quadrotor is a custom-built four-rotor rotorcraft designed and developed at the COBRA lab at UNB. It is equipped with a Hokuyo UTM-30LX scanning laser



Figure 7.6: The COBRA quadrotor equipped with a scanning laser rangefinder, an IMU, and a SODAR.

ranger, a CH Robotics UM6 IMU, and a SensComps MINI-A PB Ultrasonic ranger. The scanning laser ranger is mounted upside down for more stability during flight. Fig. 7.6 shows the quadrotor. A Gumstix Overo board is used to interface sensors and send data to a ground station.

7.3.1.2 Description

The dataset is collected in an indoor classroom environment at UNB. The classroom environment is relatively small. Its area is approximately 170 m^2 and includes two rooms. The quadrotor is guided by a remote controller. The dataset includes measurements from an inverted scanning laser ranger, a SODAR, and an IMU. The collected data was used for off-line processing to evaluate the performance of the mapping.

7.3.2 Case1: Simulation in Gazebo

The proposed solution for perception and navigation is tested in a simulated Gazebo world. Gazebo is a three-dimensional and multi-robot simulator which simulates a

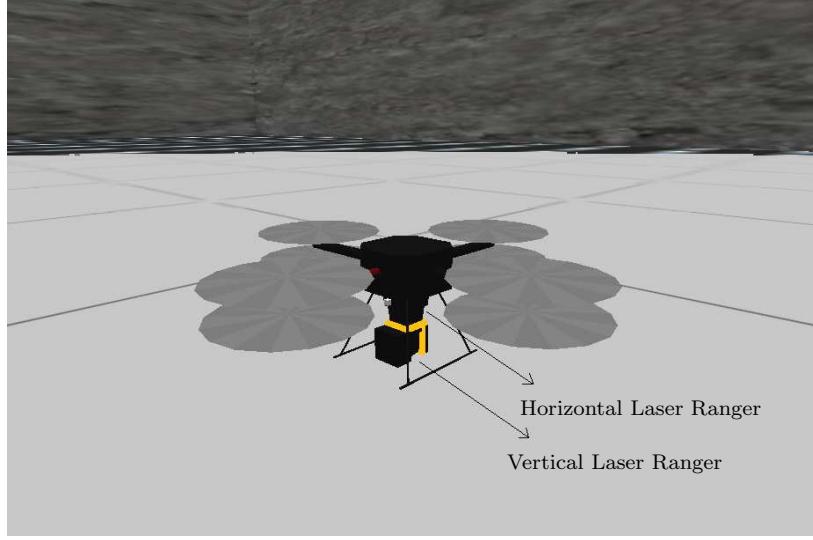


Figure 7.7: The simulated quadrotor in Gazebo with two perpendicular laser rangers.

team of robots, objects, and various sensors [168]. It also simulates gravity and interaction between objects. Gazebo has been integrated with ROS.

7.3.2.1 Simulated Flying Robot

The developed simulated quadrotor model is based on the X8 [155] rotorcraft, designed and built by Draganfly Innovations Inc. The X8 has four pairs of rotors, which provide more thrust than four rotors. The simulated robot is designed in Blender and then integrated with Gazebo. The robot is equipped with a simulated IMU, a SODAR, and two Hokuyo UTM-30LX scanning laser rangers, mounted horizontally and vertically. Fig. 7.7 shows the simulated robot. The controller of the simulated robot is explained in [55].

7.3.2.2 Description

Fig. 7.8-a shows the Gazebo world. The size of the simulated world is 32×50 metres. Seven cylinders, each with the diameter of 1 m, are placed in the world as obstacles. The distance between any two adjacent cylinders is 5 m. The height of walls and obstacles is 6 m. During flight, the robot flies at an altitude of 3 m.

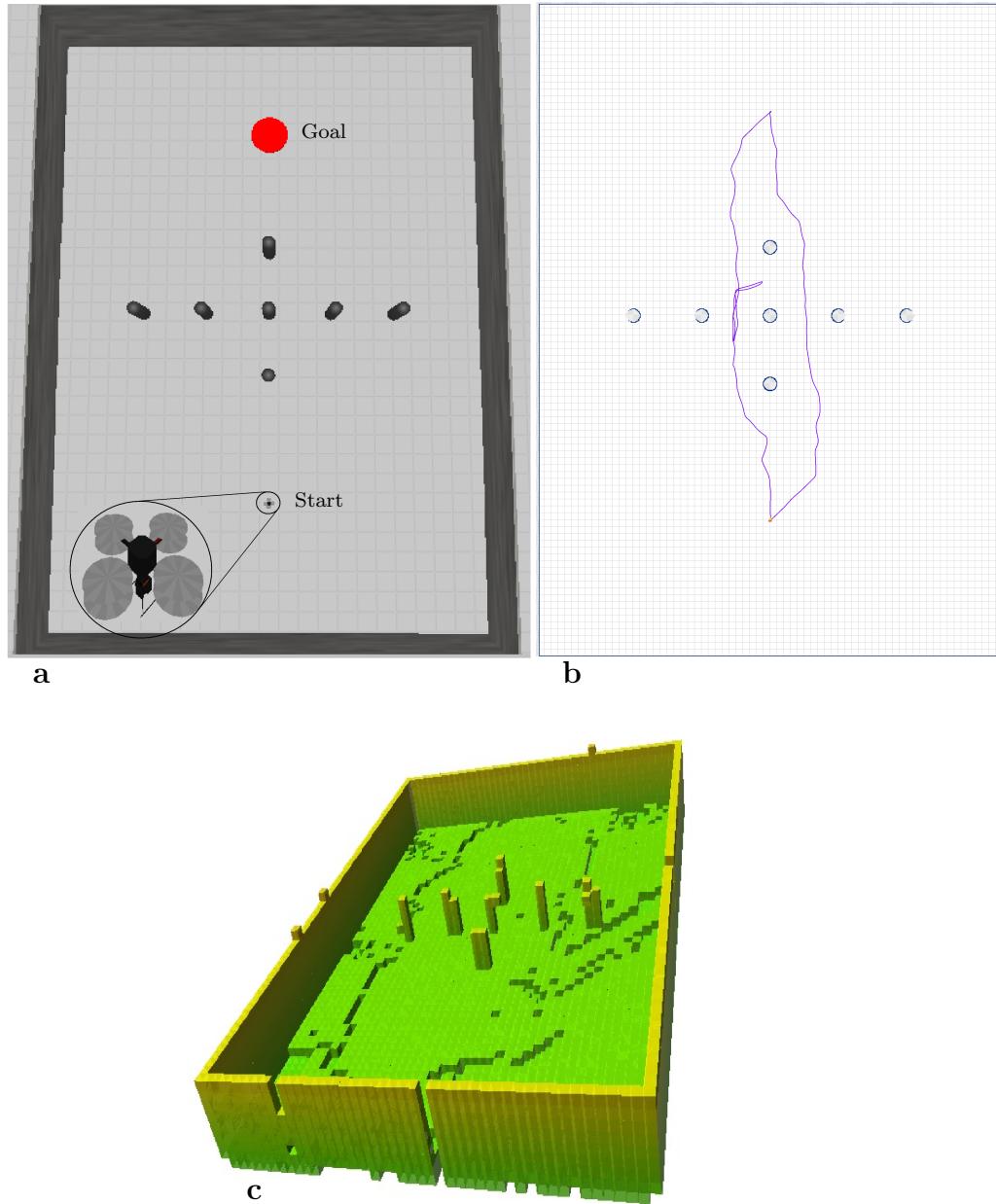


Figure 7.8: Simulation in Gazebo. **a)** Simulation environment. Size of the world is 32×50 metres. Magnified simulated X8 is shown inside a circle with two perpendicular laser rangers mounted beneath the rotorcraft. **b)** Trajectory and 2D map. **c)** 3D voxel map.

For this experiment, the mission shown in Fig. 7.2 is executed. According to this mission, first, the robot explores the world, then it goes to a goal point, and finally, it returns to the start point. The robot is initially placed at the origin of the coordinate system (Fig. 7.8-a). The goal, which it attains after the exploration, is located 30 m away, marked by a red disc.

7.3.2.3 Implementation Details

All program code is written with C++ using ROS middleware. C++ is an object-oriented programming (OOP) language. This feature of C++ is used to develop the whole project. For instance, for path planning, a class was developed that accepts a map, a start point, and a goal point. Some parameters were used to instantiate an object from the class. Given inputs, the object processes data and generates a path from the start point to the goal point, if feasible. A similar concept applies to the exploration class. It accepts a map, a start point, which is the current pose of the robot, and generates a goal point based on the exploration algorithm, explained in Algorithm 7.2.2. The goal, the map, and the start point are passed to the path planning object to generate a path. Localization and mapping are composed of several classes. These classes accept all measurements and produce a map and the pose of the robot which are used by path planning and exploration objects. The class for the mission planning takes the current task and status of the robot and determines the next task. For 3D voxel mapping and state estimation, separate classes were developed.

7.3.2.4 Results

To perform the complete mission, it took 210 seconds. In Fig. 7.8-b, the map developed by the horizontal laser ranger is shown. Fig. 7.8-c shows the voxel map of the environment, developed by the vertical laser ranger. A video of the experiment

including all states of the mission, *i.e.*, exploration, move-to-goal, and return-home, can be found in [169].

To evaluate the performance of the autonomous navigation, a few criteria are investigated. These criteria evaluate the accuracy of localization and the mission accomplishment.

7.3.2.5 Evaluating Pose Localization

Since the ground-truth position of the robot is available in the simulation, it is possible to compare the estimated position with the ground-truth position. Assume r and r_{gt} are positions of the robot through SLAM and ground-truth respectively. The same is assumed for orientations of the robot through Ω and Ω_{gt} variables. The following relations evaluate the error of the position and orientation

$$e_r = \|r - r_{gt}\|, \quad (7.8)$$

$$e_\Omega = \|\text{wrap}(\Omega - \Omega_{gt})\|, \quad (7.9)$$

where $\text{wrap}(\cdot)$ is a function which bounds the orientation so that the orientation is within the interval of $(-\pi, \pi]$. For two vectors of the same size such as p and q , the operator $\|\cdot\|$ shows the Euclidean distance defined as $\|p - q\| = \sqrt{(p - q) \cdot (p - q)}$, where the \cdot is the dot product.

7.3.2.6 Evaluating Mission Accomplishment

The implemented mission has two target points that the robot should attain. Once the exploration is finished, while the robot is moving towards the goal (the red disc), the first target point is the goal point. While the robot is returning, the second target point is the home. The coordinates of target points are known, so by comparing the position of the target points to the ground-truth position of the robot, it is possible

to calculate an error measure to evaluate targeting accuracy. If the 2D coordinates of the goal are (x_g, y_g) and the 2D ground-truth position of the robot, when it arrives at the goal, is (x_{gt}, y_{gt}) , then the target achievement error is defined as

$$e_g = \sqrt{(x_g - x_{gt})^2 + (y_g - y_{gt})^2}. \quad (7.10)$$

Fig. 7.9 shows the error of the position of the robot, e_r , during the mission. The average error is 0.0065 m. Table 7.1 summarizes the performance indices. In this table, the first row represents the average position error over the course of the mission. The second row shows the average orientation error over the course of the mission. Rows 3 and 4 show the error of the position when the robot is at the goal and start points.

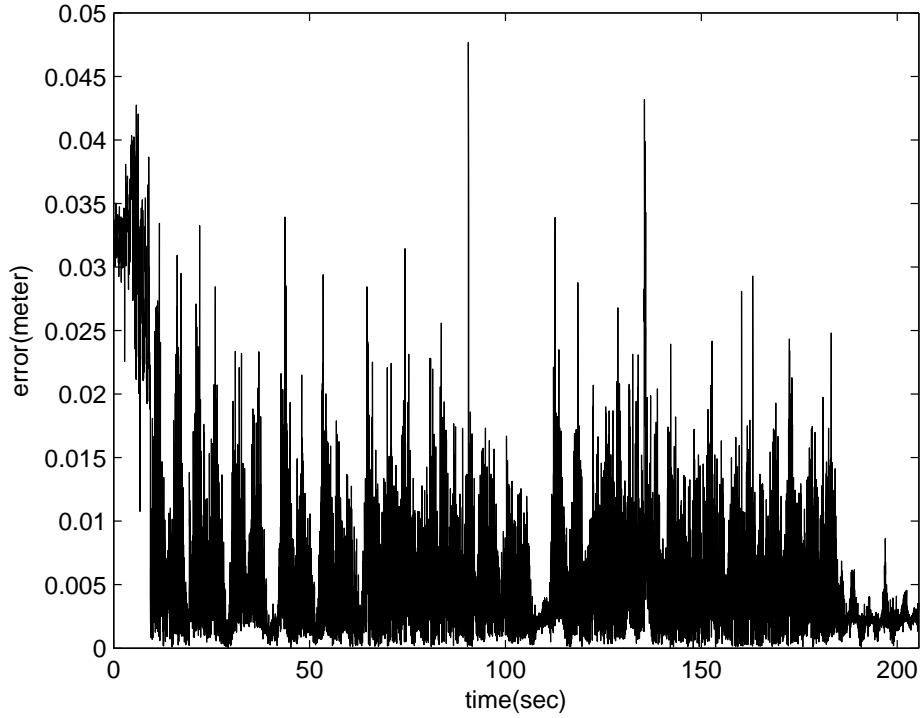


Figure 7.9: Robot position error during mission.

Table 7.1: Evaluating the experiment in Gazebo.

no	index	value
1	average position error, e_r	0.0065 m
2	average orientation error, e_Ω	0.0016 rad
3	error of achieving goal point	0.09 m
4	error of achieving start point	0.02 m

7.3.3 Case2: Unmanned Ground Robot

To make sure that the proposed solution works well in the real world, we started the experiments with a CoroBot built by CoroWare, Inc. The main objective for this test is to avoid crashing the flying robot due to unpredicted run-time problems. Furthermore, it is easier to test functionality of different components of the proposed solution on a ground robot than on a flying robot. Performing this experiment, demonstrated that the proposed solution and the sensor suite can provide efficient results.

7.3.3.1 Description

This test was performed in a room at UNB. The size of the room is approximately 5×12 metres and four garbage bins were placed in the room as obstacles. Because of the flat floor, in this case there was no rolling or pitching involved. Fig. 7.10-a shows the CoroBot in the test environment. The robot is equipped with one horizontal laser ranger, an IMU, and two encoders.

7.3.3.2 Implementation Details

All software components of this experiment, except the controller, are similar to the simulated experiment performed in Gazebo. The controller is a proportional controller that applies proper control signals to guide the robot towards waypoints.

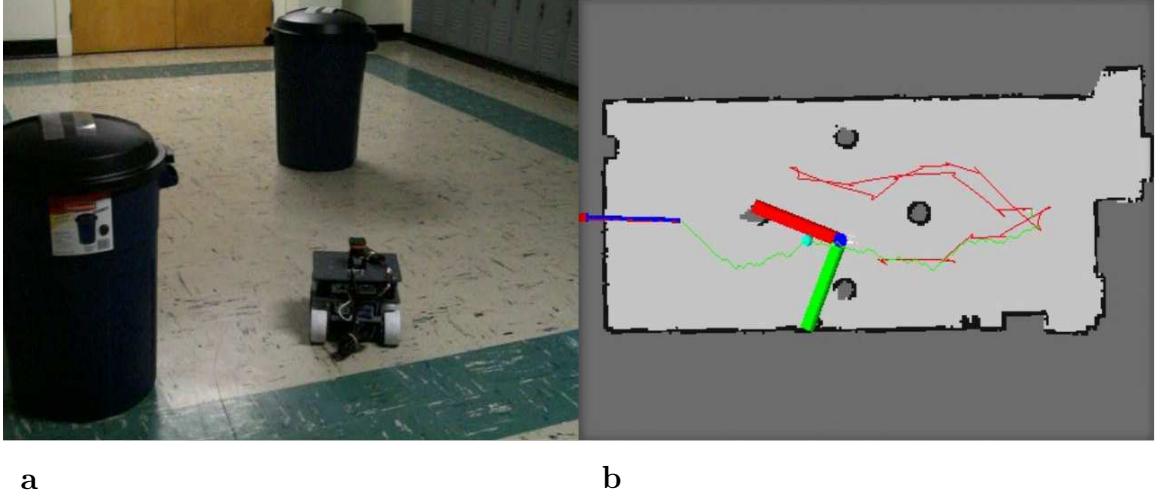


Figure 7.10: Experiment with a CoroBot robot. **a)** The robot and the experimental environment. **b)** Trajectory and 2D map. The blue arrow shows the goal, the green curve shows the planned path and the red curve shows the trajectory of the robot. The perpendicular green and red bars show the body frame coordinates. The small cyan ball on the planned path shows the waypoint to be followed by the robot.

7.3.3.3 Results

Fig. 7.10-b shows the map of the environment with the trajectory of the robot (red curve) and the planned path (green curve). A video of the experiment can be found in [169]. All states of the mission are shown in the video.

7.3.4 Case3: Quadrotor

The last real-world experiment was performed with the custom-built COBRA quadrotor. Fig. 7.11 shows the quadrotor in the test environment.

7.3.4.1 Description

The test environment is approximately 94.7 m^2 and two boxes are placed in the environment as obstacles. According to the mission plan, shown in Fig. 7.2 and starting from the point shown by a black circle in Fig. 7.11-a, first, the quadrotor explores the environment and maps all unknown places such as behind boxes. Then

it moves to a goal point, shown by a black square. Finally, it returns home, where it started the mission.

7.3.4.2 Results

The whole mission took about 320 s, with an average speed of 0.25 m/s. Fig. 7.11-b shows the robot at approximately 280 s into the test, where the robot is returning home. The green curve shows the path generated by the planner. The red curve shows the trajectory of the robot. The arrow shows the final destination of the robot which is the home. The cyan sphere shows a waypoint along the path, generated by the planner for the quadrotor. (The path, the trajectory and the waypoint are projected to the ground level for clarity of the figure.) A video of the experiment is available in [169]. The robot successfully completed the mission by exploring, mapping, and navigating through obstacles.

Fig. 7.12 shows the path of the robot for each stage of the mission. Starting from the point marked with the black circle, the blue curve shows the path of the *exploration*. The black curve shows the path of the *move-to-goal* behaviour where the goal is depicted by a black square. The green curve shows the path of the *return-home* behaviour. The red curve shows the trajectory of the robot during all three stages.

Fig. 7.13-a shows the performance of the waypoint following. The error at each time shows the distance between the current position of the robot and the given waypoint at that time. To show the process, Fig. 7.13-b shows an enlarged section of the error enclosed by a red rectangle in Fig. 7.13-a. The vertical lines indicates the times when a new waypoint was generated by the planner and the quadrotor tried to follow the waypoint. The planner updates waypoints at the approximate rate of 2 Hz and tries to keep the waypoints at a fixed distance of 0.9m from the current position of the robot, unless the robot is close to the final goal. Once the robot is closer than 0.5 m to a given goal, the planner asks the robot to hover.

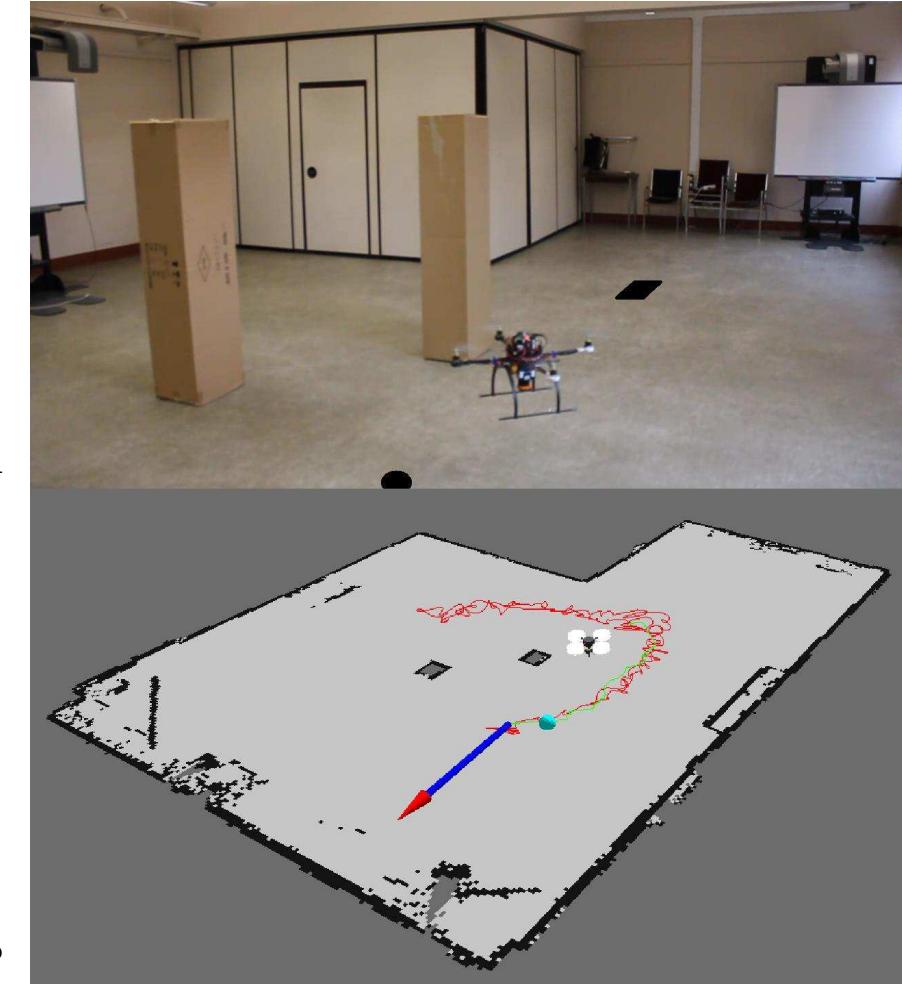


Figure 7.11: Experiment with COBRA quadrotor. This experiment, performs the three-stage mission depicted in Fig. 7.2: *exploration*, *move-to-goal*, and *return-home*. **a)** This figure shows the test environment. The robot does not know the environment in advance. The black circle marks the start point. The robot should explore the world, and then move to a goal (shown by a black square) and then return to the start point. **b)** A snapshot of the developed map and trajectory of the robot. The robot is returning home, shown by an arrow. A path has been generated and the robot is following it. The cyan sphere indicates a waypoint along the path to be followed. The robot successfully mapped the environment and navigated through obstacles and completed the mission.

Table 7.2 summarizes the performance indices for this experiment. In this experiment, the ground-truth data is not available, so evaluation of the localization error is not possible. Rows 1 and 2 show the error of the position when the robot is in the goal and start points.

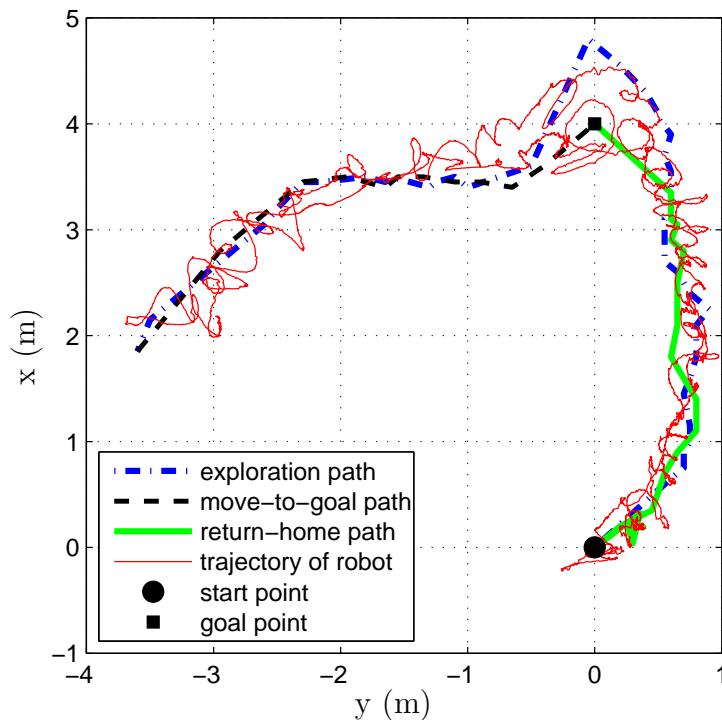


Figure 7.12: Paths and trajectory of the robot for each stage of the three-stage mission depicted in Fig. 7.2: *exploration*, *move-to-goal*, and *return-home*. The start point is shown by a circle. First the quadrotor explores the environment. The desired path for the exploration is shown in blue. Then it goes to the goal point marked by a square (*move-to-goal*). Once it reached the goal, it returned home.

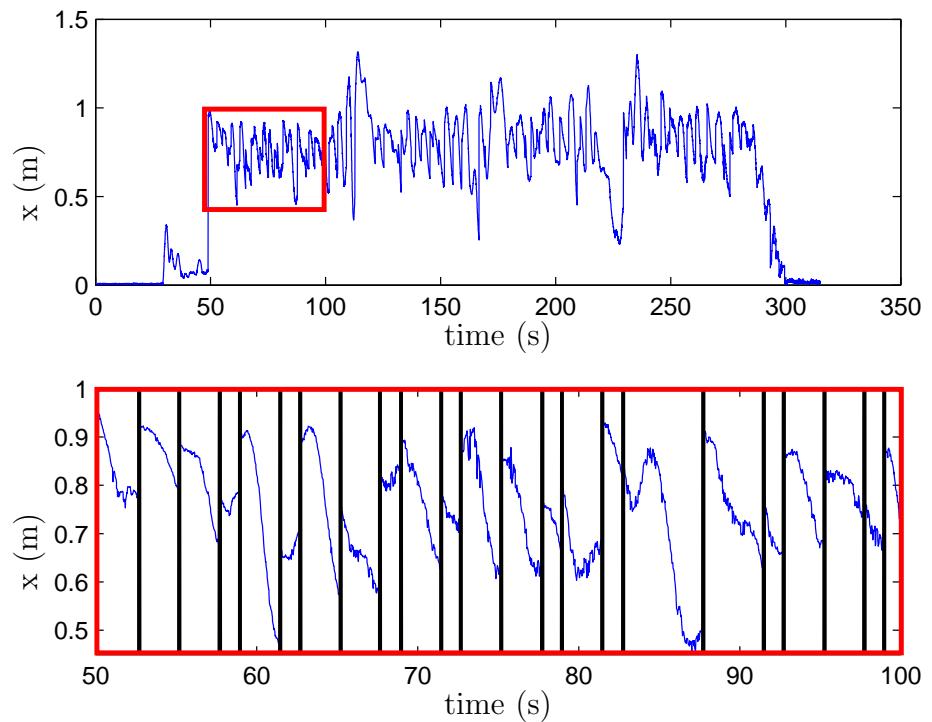


Figure 7.13: Performance of waypoint following. **a)** Waypoint following error for the whole trajectory shown in Fig. 7.12 **b)** Enlarged section of the waypoint following error enclosed in a red rectangle in Fig. 7.13-a. The vertical solid black lines indicate the times when a new waypoint is received. The quadrotor tries to follow the waypoint.

Table 7.2: Evaluating the real-world experiment with COBRA quadrotor.

no	index	value
1	error of achieving goal point	0.16 m
2	error of achieving start point	0.11 m

7.3.5 Discussion

The sequence of experiments, from the simulated quadrotor to the ground robot and then to the COBRA quadrotor, demonstrated an efficient method to construct the autonomous navigation. This approach minimizes the amount of time and resources to develop components of the system. Moreover, it shows the portability of the sensor suite and the autonomy solution from the ground robot to the quadrotor.

In simulation, as the video shows, the motion of the quadrotor was smoother than in the last experiment, performed in the real world. This is because tuning the controller in a simulated environment is easier than in a real-world environment. Also, wind effects are ignored in the simulated model. This cannot be ignored in real-world experiments, especially when the quadrotor is flying close to the ground surface.

During the last experiment, the Kinect camera was not used because of its heavy weight. However, with the IMU and the laser ranger, the 2D SLAM algorithm was able to perform localization and mapping successfully. These two sensors are minimum sensors required to perform SLAM with the quadrotor. The Kinect camera as an extra source of information may help provide more accurate results.

According to the last experiment, the errors in attaining the target points, the goal and the home, are very small. For the goal point, it is 0.16 m and for the home it is 0.11 m. This means that the quadrotor can perform specific actions while it is at a specific point. For instance, it can land on a target point such as a ground robot, or it can pick up a light object from a point and put it in a specific place.

7.4 Summary

In this work, an autonomy solution for an unmanned rotorcraft was proposed and implemented. The proposed solution tackles a few key requirements for autonomous navigation: mapping, localization, and path planning. A behaviour based mission control was proposed to plan, organize, and sequence different flight behaviours. The proposed autonomous navigation system was tested in Gazebo with a simulated flying rotorcraft and in real-world environments with an unmanned ground robot and a custom-designed quadrotor.

In the future, it would be desirable to extend the work to multiple ground and aerial robots: cooperatively exploring, mapping, localizing, and performing the mission.

Part IV

Conclusions

Simultaneous localization and mapping is a key requirement in robotic applications to perceive the environment and interact with it. Without reliable understanding of surroundings, it is not possible to perform autonomous tasks and remove human intervention. In the past chapters, this important issue was studied in two scenarios: 1) multiple ground robots, and 2) an autonomous quadrotor rotorcraft. For each of these scenarios, related problems were identified and solutions were proposed and validated in simulated and real-world experiments. This part of the thesis presents the conclusions for the multiple-robot SLAM and quadrotor projects. The conclusions are presented in separate chapters for each project. Each chapter includes a summary of the contributions followed by an outline of future work.

Chapter 8

Multiple-robot SLAM

In this chapter, the summary of the contributions with some general conclusions for multiple-robot SLAM are presented. Then future research related to the problems and the proposed methods is described.

In Part II, two general solutions for multiple-robot SLAM were studied: map merging and particle filter-based multiple-robot SLAM. For map merging, four methods were proposed. In the first method, map segmentation, using image processing techniques, featured sections of the maps were used to merge maps. In the second method, which is based on neural networks, maps were learned and merged using self-organizing maps. In the third solution, probabilistic GVD, a new topological solution was proposed. Finally, in the fourth method, Hough peak matching, map merging was performed in the Hough space. Following map merging, a particle filter-based multiple-robot SLAM algorithm was presented which proposed an efficient particle filter and considered different issues in multiple-robot SLAM. Various experiments, simulated and real-world, in different environments, demonstrated the effectiveness of the proposed solutions.

8.1 Summary of Contributions

In this section, summary of the contributions for two general proposed solutions is presented briefly.

8.1.1 Map Merging

In map merging, maps from multiple robots are merged to generate a global map. This approach has its own advantages and disadvantages which were reviewed in Chapter 4. In this thesis, four methods for map merging were proposed. The contributions of these methods are listed as follows.

- **Map segmentation:** This is a novel approach that finds the relative transformation between the robots. It has two steps, first an approximate estimate is calculated; then the results are tuned. To find the approximate transformation, first the Canny edges of the maps are smoothed by a novel smoothing method to reduce computational demand. Then the smoothed edges are filtered by the proposed histogram filter. The filter selects segments from the maps that have features. Then, the segments are matched against each other to find an approximate transformation. Once the approximate transformation is calculated, it is tuned. The rotation angle is tuned using the Radon transform and the translation is tuned utilizing a similarity index. At the end results are verified using a verification index.
- **Self-organizing maps:** This novel method also finds the relative transformation between the robots. To the best of our knowledge, in the literature, there is no similar method which uses competitive neural networks for map merging. In this approach maps are learned using self-organizing neural networks. This process reduces the dimensionality of the maps since maps are represented by a set of clusters. Then a novel approach is proposed to match clusters which is

based on the norm defined for each cluster.

- **Probabilistic generalized Voronoi diagram:** This innovative algorithm is designed to consider the probabilistic nature of the maps and preferentially match parts of the maps which have higher certainty. Using a novel approach, the GVDs of maps are extended to reflect their probabilistic properties. Then edges of the probabilistic GVDs are matched to find the relative transformation. Once the relative transformation is known, maps are fused with an entropy filter. The proposed entropy filter utilizes the additivity property of *log odds* of the occupancy grid maps and fuses the aligned maps efficiently. A new verification method based on the image entropy is proposed to verify the matching results. Moreover, in this method, the uncertainty of the relative transformation is also taken into account using a method we termed linearized uncertainty propagation. This new approach linearizes the transformation and propagates the uncertainty to the transformed maps.
- **Hough peak matching:** In this method, the transformations between the robots are found in the Hough space rather than the Euclidean space. To do this, first the maps are transformed into the Hough space. Then, in the Hough space, peak points of the Hough images are selected. The peak points represent complete and compact models of the occupancy grid maps. Then using the novel peak matching algorithm, the relative transformations are calculated. This solution is fast and robust and to the best of our knowledge, there is no similar method of matching occupancy grid maps in the literature.

8.1.2 Particle Filter-based Multiple-robot SLAM

In Chapter 5 a particle filter-based multiple-robot SLAM algorithm was presented. This method is a hybrid solution which is based on sharing raw data between the

robots and also map merging. Advantages and disadvantages of sharing raw data were described in Chapter 3. Contributions of the proposed method are listed as follows.

- **Multiple-robot SLAM with line-of-sight observations:** A new formulation for multiple-robot SLAM was derived that takes into account the line-of-sight observations between the robots. It was shown that this decreases the localization and mapping errors.
- **Particle regeneration:** A novel algorithm was proposed that propagates the uncertainty of the relative transformations on maps and poses of all robots in the team. In the proposed method, particles are regenerated based on the uncertainty of the relative transformation matrices.
- **Batch update:** To update the maps and poses, a batch update method was proposed. Once the relative transformation is known, the batch update processes all past information at once and updates the maps and poses. The main advantage of the batch update is saving time since it avoids reprocessing past data item by item. Also it saves memory by not storing all past raw data.
- **Hybrid method:** Previous map merging results were combined with particle filtering to avoid having to plan rendezvous which adds path planning challenges to multiple-robot SLAM. This advantage has been achieved by calculating the relative transformations between the robots using map merging results.
- **Complexity:** A novel proposal distribution was introduced which reduces the number of the particles significantly. The reduced complexity makes the algorithm more scalable. Moreover, by using few particles, the algorithm can even be implemented on slow computers.

8.2 Future Research

In the future, it would be desirable to extend the work to include complicated scenarios such as heterogenous ground and aerial robots or robots with different type of sensors.

These further projects can be listed as follows:

- **Multiple ground and aerial robots:** Different types of robots such as aerial and ground robots can develop maps at different altitudes. While this has the advantage of representing a more complete model of the world, it requires dealing with coordination and cooperation of heterogenous robots.
- **Heterogenous sensors:** If robots' maps are developed using different sensors such as cameras or scanning laser rangers, then a systematic approach is required to make a global map from different types of maps.
- **Updating trajectories in map merging:** map merging is not very flexible at updating trajectories of the robots. Although this problem was investigated in Chapter 5, this is still a challenging problem, primarily because there is no access to raw data from which the maps were built and the trajectories are calculated.
- **3D map merging:** All map merging methods were developed for 2D maps; however, they can be extended for 3D voxel maps. Specifically Hough peak matching and probabilistic GVD can easily be extended for 3D maps. High computational demand is the main challenge in 3D map merging.
- **Integrating multiple-robot SLAM with path planning:** It is also an interesting topic to optimize and plan paths of the robots such that there is enough overlap for map merging. Moreover, the paths can also be designed in such a way that the mapping mission finishes within a minimum required time.

Chapter 9

Rotorcraft SLAM

In this chapter, the summary of the contributions with some general conclusions in relation to the quadrotor project is presented. Then future research related to the problems is outlined.

In Part III, a complete solution for perception and navigation of autonomous quadrotors was presented. The proposed solution enables the quadrotor to fly in GPS-denied environments autonomously and achieve mission goals. Multiple tests, simulated for the quadrotor and real-world for the ground robot and COBRA quadrotor, in different environments, demonstrated the effectiveness of the proposed solution.

9.1 Summary of Contributions

The perception and navigation for autonomous rotorcraft proposes an autonomy solution for a flying robot in GPS-denied environments. The details of the contributions are listed as follows:

- **Autonomy blocks:** Main building blocks of the autonomy solution including mission control, map learning (composed of mapping, localization, and path planning), and state estimation were implemented and integrated towards au-

tonomy goals.

- **Autonomous behaviours:** At the mission level, a set of basic behaviours were defined which can be used to perform more complicated tasks.
- **Perception:** Localization and mapping were performed using view-based and feature-based SLAM algorithms. The results were fused to provide a better state estimation.
- **Path planning:** A path planning algorithm based on the wavefront algorithm was developed which guides the robot between waypoints.
- **Exploration:** The robot used the developed map to explore unknown parts of the world using the frontier algorithm. To do this, path planning, localization, and mapping were used.
- **Portability of solution:** The proposed sensor suite and autonomy solution can easily be utilized on ground robots.

9.2 Future Research

In the future, it would be desirable to extend the work to cover different aspects of autonomy, including collaboration of multiple agents and extending the boundaries of the autonomy for each robot. These further projects can be listed as follows:

- **Multiple quadrotors:** A team of multiple autonomous quadrotors can collaborate with each other to explore, map, localize, and perform missions faster and more robustly; however, performing perception and navigation by a team of quadrotors requires dealing with challenges inherent in multi-agent systems.
- **Multiple ground and aerial robots:** Extending the collaboration level to include ground robots is another interesting future undertaking. Ground robots

have the advantage that they are not limited by short flight times or payload limitations, but they have limited manoeuvrability in cluttered environments. On the other hand, quadrotors have limitations in flight time or payload, but they are not limited to the ground level. These advantages can enable a team of ground and aerial robots to be more efficient than a team of homogenous ground or aerial robots.

- **Adaptive dilation:** To navigate between points, wavefront path planning was used. Although wavefront path planning is a fast algorithm, it sometimes becomes too close to obstacles. To solve this problem, obstacles were dilated and then a path was designed. In this work, dilation radius was determined based on the size of the quadrotor. It would be desirable to change the dilation radius adaptively: if a goal point is not accessible due to a larger dilation, then the dilation radius can be reduced to its minimum size, and if there is room to increase the dilation to be further away than obstacles, then the dilation radius can be increased.
- **Path planning in 3D:** Extending path planning to 3D enables the robot to have more flexibility in changing altitude of the flight; however, to do this, it is necessary to use 3D maps which in turn requires high computational demand.
- **Multiple 2D maps:** While the quadrotor flies at different altitudes, for certain altitudes, a separate 2D map can be developed. This way the 3D world can be modelled with multiple 2D maps.
- **Fusing 2D and 3D maps:** The 2D occupancy grid map can be fused with the 3D voxel map to generate more reliable maps.
- **Exploration in 3D:** The 3D voxel map can be used for exploration instead of the 2D map. This is useful in applications such as exploring a building with

multiple floors or exploring an environment which has windows at different altitudes.

- **Complex behaviours:** The navigation stack is capable of performing more complex behaviours. These behaviours can easily be added as separate modules and recalled whenever needed. Examples of these behaviours could be following a person using face detection/recognition, following sound sources, and picking up/delivering small objects.

Bibliography

- [1] H. D. Whyte and T. Bailey, “Simultaneous localization and mapping (SLAM): Part I the essential algorithms,” *IEEE Robotics and Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.
- [2] A. Birk and S. Carpin, “Merging occupancy grid maps from multiple robots,” *Proceedings of the IEEE: Special Issue on Multi-Robot Systems*, vol. 94, no. 7, pp. 1384–1387, 2006.
- [3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: The MIT press, 2005.
- [4] A. Elfes, “Occupancy grids: A stochastic spatial representation for active robot perception,” in *Sixth Conference on Uncertainty in AI*, 1990, pp. 7–24.
- [5] S. Thrun, “Learning occupancy grids with forward sensor models,” *Autonomous Robots*, vol. 15, pp. 111–127, 2003.
- [6] B. Kuipers and Y. Byun., “A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations,” *Journal of Robotics and Autonomous Systems*, vol. 8, pp. 47–63, 1991.
- [7] R. Smith, M. Self, and P. Cheeseman, “A stochastic map for uncertain spatial relationships,” in *Fourth International Symposium of Robotics Research*, 1987, pp. 467–474.

- [8] R. Smith and P. Cheeseman, “On the representation and estimation of spatial uncertainty,” *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1987.
- [9] F. T. Ramos, J. Nieto, and H. D. White, “Recognising and modelling landmarks to close loops in outdoor SLAM,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2007, pp. 2036–2041.
- [10] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges,” in *International Joint Conference of Artificial Intelligence*, 2003, pp. 1151–1156.
- [11] S. Williams and I. Mahon, “Simultaneous localisation and mapping on the great barrier reef,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2004, pp. 1771–1776.
- [12] T. Bailey, “Mobile robot localization and mapping in extensive outdoor environments,” Ph.D. dissertation, University of Sydney, Sydney, NSW, Australia, 2000.
- [13] R. M. Eustice, H. Singh, and J. J. Leonard, “Exactly sparse delayed-state filters for view-based SLAM,” *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1100–1114, 2006.
- [14] P. Newman, “Special issue on SLAM in the field,” *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 1–165, 2007.
- [15] M. Bosse and R. Zlot, “Map matching and data association for large-scale two-dimensional laser scan-based SLAM,” *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 667–691, 2008.

- [16] J. Gutmann and K. Konolige, “Incremental mapping of large cyclic environments,” in *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, 1999, pp. 318–325.
- [17] F. Lu and E. Milios, “Globally consistent range scan alignment for environment mapping,” *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, 1997.
- [18] I. Ulrich and I. Nourbakhsh, “Appearance-based place recognition for topological localization,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2000, pp. 1023–1029.
- [19] A. Koenig, J. Kessler, and H. M. Gross, “A graph matching technique for an appearance-based, visual SLAM-approach using Rao-Blackwellized particle filters,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 1576–1581.
- [20] S. Thrun and Y. Liu, “Multi-robot SLAM with sparse extended information filers,” *Springer Tracts in Advanced Robotics*, vol. 15, pp. 254–266, 2005.
- [21] X. S. Zhou and S. I. Roumeliotis, “Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 1785–1792.
- [22] A. Gil, O. Reinoso, M. Ballesta, and J. Miguel, “Multi-robot visual SLAM using a Rao-Blackwellized particle filter,” *Robotics and Autonomous Systems*, vol. 58, pp. 68–80, 2010.
- [23] K. LeBlanc and A. Saffiotti, “Multirobot object localization: A fuzzy fusion approach,” *IEEE Transactions on Systems, Man and Cybernetics-Part B*, vol. 39, no. 5, pp. 1259–1276, 2009.

- [24] R. Aragues, J. Cortes, and C. Sagües, “Distributed consensus algorithms for merging feature-based maps with limited communication,” *Robotics and Autonomous Systems*, vol. 59, pp. 163–180, 2011.
- [25] C. M. Gifford, R. Webb, J. Bley, D. Leung, M. Calnon, J. Makarewicz, B. Banz, and A. Agah, “A novel low-cost, limited-resource approach to autonomous multi-robot exploration and mapping,” *Robotics and Autonomous Systems*, vol. 58, pp. 186–202, 2010.
- [26] A. Eliazar and R. Parr, “DP-SLAM 2.0,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2004.
- [27] W. H. Huang. and K. R. Beavers, “Topological map merging,” *The International Journal of Robotics Research*, vol. 24, no. 8, pp. 601–613, 2005.
- [28] A. Howard, “Multi-robot simultaneous localization and mapping using particle filters,” *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243–1256, 2006.
- [29] A. Howard, L. Parker, and G. Sukhatme, “Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection,” *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 431–447, 2006.
- [30] S. Carpin, A. Birk, and V. Jucikas, “On map merging,” *Robotics and Autonomous Systems*, vol. 53, pp. 1–14, 2005.
- [31] S. Carpin, “Fast and accurate map merging for multi-robot systems,” *Autonomous Robots*, vol. 25, pp. 305–316, 2008.

- [32] K. YK Leung, T. Barfoot, and H. HT Liu, “Decentralized localization of sparsely-communicating robot networks: A centralized-equivalent approach,” *IEEE Transactions on Robotics*, vol. 26, pp. 62–77, 2010.
- [33] ——, “Decentralized cooperative SLAM for sparsely-communicating robot networks: A centralized-equivalent approach,” *Journal of Intelligent and Robotic Systems*, vol. 66, pp. 321–342, 2012.
- [34] Y. Bar-Shalom, “Update with out-of-sequence measurements in tracking: exact solution,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, pp. 769–777, 2002.
- [35] Y. Bar-Shalom, H. Chen, and M. Mallick, “One-step solution for the multi-step out-of-sequence-measurement problem in tracking,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, pp. 27–37, 2004.
- [36] A. Howard, M. Mataric, and G. Sukhatme, “Putting the ‘I’ in ‘team’: an ego-centric approach to cooperative localization,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2003, pp. 868–874.
- [37] S. Thrun, M. Diel, and D. Hahnel, “Scan alignment and 3-D surface modeling with a helicopter platform,” *Field and Service Robotics*, vol. 24, pp. 287–297, 2006.
- [38] S. Grzonka, G. Grisetti, and W. Burgard, “Towards a navigation system for autonomous indoor flying,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2009, pp. 2878–2883.
- [39] ——, “A fully autonomous indoor quadrotor,” *IEEE Transaction on Robotics*, vol. 28, no. 1, pp. 90–100, 2012.

- [40] A. Bachrach, R. He, and N. Roy, “Autonomous flight in unknown indoor environments,” *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 217–228, 2009.
- [41] E. Olson, J. Leonard, and S. Teller, “Fast iterative alignment of pose graphs with poor initial estimates,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2006, pp. 2262–2269.
- [42] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Proceedings of the 9th European Conference on Computer Vision*.
- [43] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with Rao-Blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, 2007.
- [44] B. Yamauchi, A. Schultz, and W. Adams, “Integrating exploration and localization for mobile robots,” *Autonomous Robots*, 1999.
- [45] R. He, S. Prentice, and N. Roy, “Planning in information space for a quadrotor helicopter in a GPS-denied environment,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2008, pp. 1814–1820.
- [46] I. Dryanovski, W. Morris, and J. Xiao, “An open-source pose estimation system for micro-air vehicles,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4449–4454.
- [47] A. Censi, “An ICP variant using a point-to-line metric,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2008.

- [48] S. Kohlbrecher, O. v. Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable SLAM system with full 3D motion estimation,” in *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011, pp. 155–160.
- [49] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. Grix, F. Ruess, M. Suppa, and D. Burschka, “Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 46–56, 2012.
- [50] G. M. Hoffmann, S. L. Wasl, and C. J. Tomlin, “Quadrotor helicopter trajectory tracking control,” in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2008.
- [51] S. Koenig and M. Likhachev, “Fast replanning for navigation in unknown terrain,” *IEEE Transaction on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.
- [52] M. Vitus, V. Pradeep, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, “Tunnel-MILP: Path planning with sequential convex polytopes,” in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2008.
- [53] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, “The GRASP multiple micro-UAV testbed,” *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.
- [54] S. Bouabdallah, “Design and control of quadrotors with application to autonomous flying,” Ph.D. dissertation, Ecole Polytechnique Federale De Lausanne (EPFL), Lausanne, Switzerland, 2007.
- [55] A. Nagaty, S. Saeedi, C. Thibault, M. Seto, and H. Li, “Control and navigation framework for quadrotor helicopters,” *Journal of Intelligent and Robotic Systems*, vol. 70, no. 1-4, pp. 1–12, 2013.

- [56] Hokuyo, “UTM-30LX,” accessed 22 July 2013. [Online]. Available: http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html
- [57] SensComp, “MINI-A ultrasonic sensor,” accessed 22 July 2013. [Online]. Available: <http://www.senscomp.com/pdfs/mini-a-ultrasonic-sensor.pdf>
- [58] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, 4th ed. New York City, NY, USA: McGraw Hill, 2002.
- [59] D. Frank and K. Michael, “Square root SAM: Simultaneous location and mapping via square root information smoothing,” *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181 – 1203, 2006.
- [60] B. Ristic, S. Arulampalm, and N. J. Gordon, *Beyond the Kalman Filter*. London, UK: Artech House.
- [61] A. Doucet, N. d. Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. Berlin, Germany: Springer-Verlag, 2001.
- [62] J. Hawkins, *On Intelligence*. New York City, NY, USA: Times Books, 2004.
- [63] J. L. Gould, “Honey bee cognition,” *Cognition*, vol. 37, pp. 83–103, 1990.
- [64] A. W. Siegel and S. H. White, “The development of spatial representations of large-scale environments,” *Advances in Child Development and Behavior*, vol. 10, pp. 9–55, 1975.
- [65] P. Beeson, J. Modayil, and B. Kuipers, “Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy,” *The International Journal of Robotics Research*, vol. 29, no. 4, pp. 428–459, 2010.
- [66] E. R. Davies, *Machine Vision, Theory, Algorithms, Practicalities, 3rd Edition*. Burlington, MA, USA: Morgan Kaufmann, 2005.

- [67] R. Smith, M. Self, and P. Cheeseman, “A stochastic map for uncertain spatial relationships,” in *Fourth International Symposium of Robotics Research*, 1987, pp. 467–474.
- [68] J. Guivant and E. Nebot, “Optimization of the simultaneous localization and map-building algorithm and real-time implementation,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001.
- [69] A. J. Davison and D. W. Murray, “Simultaneous localization and map-building using active vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 865–880, 2002.
- [70] L. Jaulin, “A nonlinear set membership approach for the localization and map building of underwater robots,” *IEEE Transactions on Robotics*, vol. 25, no. 1, pp. 88 –98, 2009.
- [71] P. M. Newman, J. J. Leonard, and R. J. Rikoski, “Towards constant-time SLAM on an autonomous underwater vehicle using synthetic aperture sonar,” in *Proceedings of the Eleventh International Symposium on Robotics Research*, 2003, pp. 409–420.
- [72] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, “An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 206–211.
- [73] F. Lu and E. E. Milios, “Robot pose estimation in unknown environments by matching 2D range scans,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994, pp. 935–938.
- [74] J. Nieto, T. Bailey, and E. Nebot, “Recursive scan-matching SLAM,” *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 39–49, 2007.

- [75] L. Paull, S. Saeedi, M. Seto, and H. Li, “Auv navigation and localization - a review,” *IEEE Journal of Oceanic Engineering*, In Press.
- [76] M. Bosse and R. Zlot, “Map matching and data association for large-scale two-dimensional laser scan-based SLAM,” *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 667–691, 2008.
- [77] M. Cummins and P. Newman, “Appearance-only SLAM at large scale with FAB-MAP 2.0,” *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.
- [78] A. Kawewong, N. Tongprasit, S. Tangruamsub, and O. Hasegawa, “Online and incremental appearance-based SLAM in highly dynamic environments,” *The International Journal of Robotics Research*, vol. 30, no. 1, pp. 33–55, 2010.
- [79] D. Hähnel, W. Burgard, and S. Thrun, “Learning compact 3D models of indoor and outdoor environments with a mobile robot,” *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 15–27, 2003.
- [80] J. Weingarten and R. Siegwart, “3D SLAM using planar segments,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 3062–3067.
- [81] K. Pathak, A. Birk, N. Vaskevicius, M. Pfingsthorn, S. Schwertfeger, and J. Poppinga, “Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation,” *Journal of Field Robotics*, vol. 27, no. 1, pp. 52–84, 2010.
- [82] S. Thrun, C. Martin, Y. Liu, D. Hahnel, R. Emery-Montemerlo, D. Chakrabarti, and W. Burgard, “A real-time expectation-maximization algorithm for acquiring multiplanar maps of indoor environments with mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 433–443, 2004.

- [83] R. Smith, M. Self, and P. Cheeseman, “A stochastic map for uncertain spatial relationships,” in *Fourth International Symposium of Robotics Research*, 1987, pp. 467–474.
- [84] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, “Simultaneous localization and mapping with sparse extended information filters,” *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.
- [85] M. R. Walter, R. M. Eustice, and J. J. Leonard, “Exactly sparse extended information filters for feature-based SLAM,” *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, 2007.
- [86] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A factored solution to the simultaneous localization and mapping problem,” in *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.
- [87] A. Eliazar and R. Parr, “DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks,” in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 1135–1142.
- [88] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, “Hierarchical optimization on manifolds for online 2D and 3D mapping,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010.
- [89] A. Howard, M. Mataric, and G. Sukhatme, “Relaxation on a mesh: a formalism for generalized localization,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, 2001, pp. 1055–1060.

- [90] U. Frese, P. Larsson, and T. Duckett, “A multilevel relaxation algorithm for simultaneous localization and mapping,” *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 196–207, 2005.
- [91] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Fast incremental smoothing and mapping with efficient data association,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2007, pp. 1670–1677.
- [92] G. Grisetti, C. Stachniss, and W. Burgard, “Nonlinear constraint network optimization for efficient map learning,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 428–439, 2009.
- [93] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based SLAM,” *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.
- [94] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg, “Hierarchical optimization on manifolds for online 2D and 3D mapping,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010.
- [95] E. Olson, “Robust and efficient robotic mapping,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008.
- [96] C. Hertzberg, “A framework for sparse, non-linear least squares problems on manifolds,” Master’s thesis, University of Bremen, Bremen, Germany, 2008.
- [97] C. Estrada, J. Neira, and J. Tardos, “Hierarchical SLAM: Real-time accurate mapping of large environments,” *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

- [98] K. Ni, D. Steedly, and F. Dellaert, “Tectonic SAM: Exact, out-of-core, submap-based SLAM,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2007, pp. 1678 – 1685.
- [99] A. Chatterjee, “Differential evolution tuned fuzzy supervisor adapted extended Kalman filtering for SLAM problems in mobile robots,” *Robotica*, vol. 27, no. 3, pp. 411–423, 2009.
- [100] G. Wyeth and M. Milford, “Spatial cognition for robots,” *IEEE Robotics and Automation Magazine*, vol. 16, no. 3, pp. 24–32, 2009.
- [101] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, “Simultaneous localization and mapping with sparse extended information filters,” *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.
- [102] Virginia Vassilevska Williams, “Multiplying matrices faster than Coppersmith-Winograd,” in *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, 2012, pp. 887–898.
- [103] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments.” in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2010.
- [104] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, “Real-time 3D visual SLAM with a hand-held RGB-D camera,” in *Proceedings of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, 2011.
- [105] A. Segal, D. Haehnel, and S. Thrun, “Generalized ICP,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2009.

- [106] S. Thrun, “A probabilistic on-line mapping algorithm for teams of mobile robots,” *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.
- [107] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, “Multiple relative pose graphs for robust cooperative mapping,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010, pp. 3185–3192.
- [108] P. Mahalanobis, “On the generalised distance in statistics,” in *Proceedings National Institute of Science, India*, vol. 2, no. 1, 1936, pp. 49–55.
- [109] L. Carlone, M. K. Ng, J. Du, B. Bona, and M. Indri, “Rao-Blackwellized particle filters multi robot SLAM with unknown initial correspondences and limited communication,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010, pp. 243–249.
- [110] L. Andersson and J. Nygards, “C-SAM: Multi-robot SLAM using square root information smoothing,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2008, pp. 2798–2805.
- [111] A. Cunningham and F. Dellaert, “Large-scale experimental design for decentralized SLAM,” in *Proceedings of SPIE, Unmanned Systems Technology XIV*, vol. 8387, 2012.
- [112] S. Williams, G. Dissanayake, and H. Durrant-Whyte, “Towards multi-vehicle simultaneous localisation and mapping,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, vol. 3, 2002, pp. 2743–2748.

- [113] A. Howard, G. Sukhatme, and M. Mataric, “Multirobot simultaneous localization and mapping using manifold representations,” *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1360–1369, 2006.
- [114] S. Carpin, “Merging maps via Hough transform,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 1878–1883.
- [115] J. Radon, “On the determination of functions from their integral values along certain manifolds,” *IEEE Transactions on Medical Imaging*, vol. 5, no. 4, pp. 170–176, 1986.
- [116] M. Wang and C.-H. Lai, *A Concise Introduction to Image Processing*. Boca Raton, FL, USA: CRC Press, 2008.
- [117] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, “Engineering applications of the self-organizing map,” *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1358–1384, 1996.
- [118] T. Kohonen, “Fast evolutionary learning with batch-type self-organizing maps,” *Neural Processing Letters*, vol. 9, pp. 153–162, 1999.
- [119] P. Somervuo and T. Kohonen, “Self-organizing maps and learning vector quantization for feature sequences,” *Neural Processing Letters*, vol. 10, pp. 151–159, 1999.
- [120] T. Kohonen, *Self-Organizing and Associative Memory*. Berlin, Germany: Springer-Verlag, 1987.
- [121] S. Saeedi, L. Paull, M. Trentini, and H. Li, “Multiple robot simultaneous localization and mapping,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 853–858.

- [122] F. Lu and E. Milios, “Robot pose estimation in unknown environments by matching 2d range scans,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994, pp. 935–938.
- [123] G. A. F. Seber, *Multivariate Observations*. Hoboken, NJ, USA: John Wiley and Sons Inc., 2004.
- [124] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub, “Interpreting patterns of gene expression with self-organizing maps,” in *Proceedings of the National Academy of Science*, vol. 96, 1999, pp. 2907–2912.
- [125] H. Choset and K. Nagatani, “Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125–137, 2001.
- [126] H. Choset, S. Walker, K. Eiamsa-Ard, and J. Burdick, “Sensor-based exploration: incremental construction of the hierarchical generalized Voronoi graph,” *The International Journal of Robotics Research*, vol. 19, no. 2, pp. 125–148, 2000.
- [127] J. O. Wallgrum, “Voronoi graph matching for robot localization and mapping,” *Transactions on computational science IX*, vol. 6290, pp. 76–108, 2010.
- [128] A. Ranganathan, E. Menegatti, and F. Dellaert, “Bayesian inference in the space of topological maps,” *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 92–107, 2006.
- [129] S. Friedman, H. Pasula, and D. Fox, “Voronoi random fields: Extracting the topological structure of indoor environments via place labeling,” in *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2007, pp. 2109–2114.

- [130] D. Silver, D. Ferguson, A. Moms, and S. Thayer, “Feature extraction for topological mine maps,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 773–779.
- [131] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*. Cambridge, MA, USA: The MIT Press, 2005.
- [132] H. Blum, “A transformation for extracting new descriptors of shape,” *Models for the Perception of Speech and Visual Form*, pp. 362–381, 1967.
- [133] B. Lau, C. Sprunk, and W. Burgard, “Improved updating of Euclidean distance maps and Voronoi diagrams,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 281–286.
- [134] P. Beeson, N. K. Jong, and B. Kuipers, “Towards autonomous topological place detection using the extended Voronoi graph,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2004, pp. 4373–4379.
- [135] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers, “Position estimation for mobile robots in dynamic environments,” in *Proceedings of the American Association for Artificial Intelligence (AAAI)*, 1998.
- [136] A. Howard and N. Roy, “The robotics data set repository (Radish),” 2003, accessed 19 July 2013. [Online]. Available: <http://radish.sourceforge.net/>
- [137] R. Vaughan, B. Gerkey, and A. Howard, “The Player/Stage project: Tools for multi-robot and distributed sensor systems,” in *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2003, pp. 317–323.

- [138] V. Nguyen, D. Fox, A. Martinellii, N. Tomatis, and R. Siegwart, “A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 1929–1934.
- [139] A. Harati and R. Siegwart, “A new approach to segmentation of 2D range scans into linear regions,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 2083–2088.
- [140] L. Iocchi and D. Nardi, “Hough localization for mobile robots in polygonal environments,” *Robotics and Autonomous Systems*, vol. 40, pp. 43–58, 2002.
- [141] P. T. Boggs, “A new algorithm for the chebyshev solution of overdetermined linear systems,” *Mathematics of Computation*, vol. 28, no. 125, pp. 203–217, 1974.
- [142] Willow Garage, “Robot Operating System.” [Online]. Available: <http://www.ros.org/>
- [143] S. Chunhavittayatera, O. Chitsobhuk, and K. Tongprasert, “Image registration using Hough transform and phase correlation,” in *The 8th International Conference on Advanced Communication Technology*, 2006, pp. 973–977.
- [144] A. Censi, L. Iocchi, and G. Grisetti, “Scan matching in the Hough domain,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2005, pp. 2739–2744.
- [145] L. Zhongke, Y. Xiaohui, and W. Lenan, “Image registration based on Hough transform and phase correlation,” in *Proceedings of the IEEE International Conference on Neural Network and Signal Processing*, 2003, pp. 956–959.

- [146] H. Onishi and H. Suzuki, “Detection of rotation and parallel translation using Hough and Fourier transforms,” in *Proceedings of the IEEE International Conference on Image Processing*, 1996, pp. 827–830.
- [147] T. A. Vidal-Calleja, C. Berger, J. Sol, and S. Lacroix, “Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain,” *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 654 – 674, 2011.
- [148] J. Gordon Leishman, *Principles of Helicopter Aerodynamics (Cambridge Aerospace Series)*. Cambridge, UK: Cambridge University Press, 2006.
- [149] P. Castillo Garcia, R. Lozano, and A. E. Dzul, *Modelling and Control of Mini-Flying Machines*. Berlin, Germany: Springer, 2005.
- [150] Wikipedia, “Breguet-Richet Gyroplane,” accessed 19 July 2013. [Online]. Available: http://en.wikipedia.org/wiki/Breguet-Richet_Gyroplane
- [151] ——, “De Bothezat flying octopus,” accessed 19 July 2013. [Online]. Available: http://en.wikipedia.org/wiki/George_de_Bothezat#cite_note-MI176-3
- [152] DIY Drones, “ArduCopter,” accessed 19 July 2013. [Online]. Available: <http://www.arducopter.co.uk/>
- [153] Parrot, “Parrot AR.Drone,” accessed 19 July 2013. [Online]. Available: <http://ardrone2.parrot.com/usa/>
- [154] Aeryon, “Aeryon Scout,” accessed 19 July 2013. [Online]. Available: <http://www.aeryon.com/products/avs/aeryon-scout.html>
- [155] Draganfly, “DraganFlyer X8,” accessed 19 July 2013. [Online]. Available: <http://www.draganfly.com/uav-helicopter/draganflyer-x8/>

- [156] I. Palunko, P. Cruz, and R. Fierro, “Agile load transportation : Safe and efficient load manipulation with aerial robots,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 69–79, 2012.
- [157] P. McKerrow, “Modelling the Draganflyer four-rotor helicopter,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, vol. 4, 2004, pp. 3596–3601.
- [158] S. Bouabdallah, A. Noth, and R. Siegwart, “PID vs LQ control techniques applied to an indoor micro quadrotor,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2451–2456.
- [159] R. Mahony, V. Kumar, and P. Corke, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.
- [160] C. Stachniss, *Robotic Mapping And Exploration*. Berlin, Germany: Springer Tracts in Advanced Robotics, 2009, vol. 55.
- [161] R. A. Brooks, “Elephants don’t play chess,” *Robotics and Autonomous Systems*, vol. 6, pp. 3–15, 1990.
- [162] M. J. Mataric, “Behavior-based control: Examples from navigation, learning, and group behavior,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 323–336, 1997.
- [163] R. D’Andrea, “Quadrocopter pole acrobatics,” accessed 19 July 2013. [Online]. Available: <http://www.flyingmachinearena.org/>
- [164] K. Konolige, “Projected texture stereos,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010.

- [165] MESA Imaging, “3D time of flight cameras,” accessed 19 July 2013. [Online]. Available: <http://www.mesa-imaging.ch/>
- [166] PrimeSense, “Kinect,” accessed 19 July 2013. [Online]. Available: <http://www.primesense.com/casestudies/kinect/>
- [167] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, “octoMap: A probabilistic, flexible, and compact 3D map representation for robotic system,” in *Proceedings of the IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2010.
- [168] N. Koenig, J. Hsu, M. Dolha, and A. Howard, “Gazebo,” accessed 19 July 2013. [Online]. Available: <http://gazebosim.org/>
- [169] S. Saeedi, A. Nagaty, C. Thibault, M. Trentini, and H. Li, “Perception and navigation of autonomous rotorcraft,” accessed 19 July 2013. [Online]. Available: <http://www.ece.unb.ca/COBRA/quadrotor.htm>

Vita

Candidate's full name:
Sajad Saeedi Gharahbolagh

2009 - Present Ph.D.
Electrical and Computer Engineering
University of New Brunswick
Advisors: Dr. Howard Li
Thesis Title: "Multiple Robot Simultaneous Localization and Mapping"

2001 - 2004 M.Sc.E
Electrical Engineering
Tarbiat Modares University
Advisor: Dr. M.T. Beheshti H.
Thesis Title: "Variable Structure Control for Congestion Control in Differentiated Services Networks"

1997 - 2001 B.Sc.E
Electrical Engineering
K.N.Toosi University of Technology

Journal Articles

- [J1] **Sajad Saeedi**, Amr Nagaty, Carl Thibault, Michael Trentini, Howard Li. "Perception and Navigation for Autonomous Rotorcraft in GPS-denied Environments." *International Journal of Robotics and Automation*, conditionally accepted.
- [J2] **Sajad Saeedi**, Liam Paull, Michael Trentini, Mae Seto and Howard Li. "Map Merging for Multiple Robots Using Hough Peak Matching." *Robotics and Autonomous Systems*, 62(10), pp. 1408-1424, 2014.
- [J3] **Sajad Saeedi**, Liam Paull, Michael Trentini, and Howard Li. "Group Mapping: A Topological Approach to Map Merging for Multiple Robots." *IEEE Robotics and Automation Magazine*, 21(2), pp. 60-72. 2014.

- [J4] **Sajad Saeedi**, Liam Paull, Michael Trentini, and Howard Li. “Occupancy Grid Map Merging for Multiple-robot Simultaneous Localization and Mapping.” *International Journal of Robotics and Automation*, 30(2), pp. 1-9, 2015 (in press).
- [J5] **Sajad Saeedi**, Liam Paull, Mike Trentini, Howard Li. “Neural Network-based Multiple Robot Simultaneous Localization and Mapping.” *IEEE Transactions on Neural Networks*, 22(12), pp. 2376 - 2387, 2012.
- [J6] Amr Nagaty, **Sajad Saeedi**, Carl Thibault, Mae Seto, Howard Li. “Control and Navigation Framework for Quadrotor Helicopters.” *Journal of Intelligent and Robotic Systems*, 70(1-4), pp. 1-12, 2013.
- [J7] Liam Paull, **Sajad Saeedi**, Mae Seto, Howard Li. “AUV Navigation and Localization - A Review.” *IEEE Journal of Oceanic Engineering*, 39(1), pp. 131-149, 2014.
- [J8] Liam Paull, **Sajad Saeedi**, Mae Seto, Howard Li. “Sensor-Driven Online Coverage Planning for Autonomous Underwater Vehicles.” *IEEE/ASME Transactions on Mechatronics*, 18(6), pp. 1827-1838, 2013.

Book Chapters

- [B1] Liam Paull, **Sajad Saeedi**, Howard Li. “Path Planning for Autonomous Underwater Vehicles.” in *Autonomy for Marine Robots*. Springer 2012. Editor: Dr. Mae Seto. pp. 177-224.
- [B2] M.L. Seto, L. Paull, **S. Saeedi**. “Introduction to Autonomy for Marine Robots.” in *Autonomy for Marine Robots*. Springer 2012. Editor:: Dr. Mae Seto.

Conference

- [C1] **Sajad Saeedi**, Amr Nagaty, Carl Thibault, Michael Trentini, Howard Li. “Perception and Navigation for Autonomous Rotorcraft.” *International Conference on Intelligent Unmanned Systems (ICIUS)*, Montreal, Quebec, Canada, Sep 29 - Oct 1, 2014.
- [C2] **Sajad Saeedi**, Liam Paull, Michael Trentini, Mae Seto, Howard Li. “Map Merging Using Hough Peak Matching.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 4683-4688, 2012.
- [C3] **Sajad Saeedi**, Liam Paull, Michael Trentini, Mae Seto, Howard Li. “Efficient Map Merging Using a Probabilistic Generalized Voronoi Diagram.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 4419-4424, 2012.
- [C4] **Sajad Saeedi**, Liam Paull, Michael Trentini, Howard Li. “Neural Network-based Multiple Robot Simultaneous Localization and Mapping.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 880-885, 2011.
- [C5] **Sajad Saeedi**, Liam Paull, Michael Trentini, Howard Li. “Multiple Robot Simultaneous Localization and Mapping.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 853-888, 2011.

- [C6] Amr Nagaty, **Sajad Saeedi**, Carl Thibault, Mae Seto and Howard Li, “Control and Navigation Framework for Quadrotor Helicopters.” *Proceedings of the International Conference On Unmanned Aircraft Systems (ICUAS)*. 2012.
- [C7] Liam Paull, **Sajad Saeedi**, Mae Seto, Howard Li. “Sensor Driven Online Coverage Planning for Autonomous Underwater Vehicles.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012.
- [C8] Liam Paull, Gaetan Severac, Guilherme V. Raffo, Julian M. Angel, Harold Boley, Maki K. Habib, Bao Nguyen, Veera R. S. Kumar, **Sajad Saeedi**, Ricardo Sanz, Mae Seto, Aleksandar Stefanovski, Michael Trentini, Howard Li. “Towards An Ontology for Autonomous Robots.” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012.
- [C9] Liam Paull, **Sajad Saeedi**, Mae Seto, Howard Li. “A Multi-Agent Framework with MOOS-IvP for Autonomous Underwater Vehicles with Sidescan Sonar Sensors.” *International Conference on Autonomous and Intelligent Systems*. pp. 41-50, 2011.
- [C10] Liam Paull, **Sajad Saeedi**, Howard Li, Vincent Myers. “An Information Gain Based Adaptive Path Planning Method for an Autonomous Underwater Vehicle Using Sidescan Sonar.” *IEEE Conference on Automation Science and Engineering (CASE)*. pp. 835-840, 2010.