



HOKKAIDO UNIVERSITY

Title	Probabilistic Approaches and Algorithms for Indoor Robot Mapping in Structured Environments
Author(s)	Ravankar, Ankit
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第12021号
Issue Date	2015-09-25
DOI	https://doi.org/10.14943/doctoral.k12021
Doc URL	https://hdl.handle.net/2115/59927
Type	doctoral thesis
File Information	Ravankar_Ankit.pdf



Probabilistic Approaches and Algorithms for Indoor Robot Mapping in Structured Environments



Ankit A. Ravankar

Graduate School of Engineering
Hokkaido University

This dissertation is submitted for the degree of
Doctor of Philosophy

I would like to dedicate this thesis to my loving parents . . .

The thesis titled

*Probabilistic Approaches and Algorithms for Indoor Robot Mapping in
Structured Environments*

by

Ankit A. Ravankar

is reviewed and approved by:

Main referee

Prof. Yukinori Kobayashi

Prof. Shunichi Kaneko

Prof. Itsuro Kajiwara

Prof. Toshiro Ohashi

Assoc. Prof. Takanori Emaru



August 2015

Acknowledgements

This thesis work would not have been possible without the kind help and support of many people. The 5 years at Hokkaido University and specifically inside the laboratory of Robotics and Dynamics has been nothing sort of amazing. Throughout these years I have learned important lessons that have helped me grow into the person I am today. I am heartily thankful to my supervisor, Professor Yukinori Kobayashi, for accepting me as a student and whose encouragement, supervision and support from the preliminary to the concluding level enabled me to develop an understanding of the subject. He is always ready for discussion and keeps giving very important suggestions for improvements. His fruitful suggestions not just in research and academic areas, but to become a good researcher are something I will remember and cherish for the rest of my life. He provided me ample opportunities to learn more about robotics and always encouraged to learn "step by step".

I would specially like to thank Associate Professor Yohei Hoshino who is now with the Kitami Institute of Technology. I thank him for the lengthy discussions and his valuable suggestions which were very useful for the completion of this thesis work. I would like to thank Associate Professor Takanori Emaru for his supervision, guidance and help at every step of my research. He has been always ready for discussion and to suggest any improvements. He has always helped to make sure that the research work is not hindered due to the lack of equipment and has always provided with all the tools and resources whenever they were needed. I want to thank him for the numerous conferences he accompanied me and the discussion we had.

I express my sincere gratitude to the examination committee for giving critical and important suggestions to improve the thesis work. Specially I would like to sincerely thank Professor Shunichi Kaneko who mentored me during my PhD studies. Throughout the course Professor Kaneko provided important insight into improving the research quality. His positive attitude and constant encouragement to do quality research has helped me a lot to finish this thesis work.

I would like to thank MEXT (Ministry of Education, Culture, Sports, Science and Technology) Japan, which provided me the scholarship (Mombusho) to support my post graduate studies in Japan for last 5 years. I am also thankful to the staff members of Graduate school of Engineering, Hokkaido University for helping in my day-to-day life in Japan. I would like to thank the e^3 (English Engineering Education Program) international student association of which I was a part for so many years. Thanks to the e^3 staff for their consistent support for all activities related to academic life. To all my lab mates for helping me with the experiments, discussions and Japanese language. A special thanks to Jixin Lv for his help and fruitful discussions during my research experiments and giving valuable contribution to improve the results.

I offer my regards to my parents who worked so hard all their life to give us the best of education and provided us all means of comfort. For the hardships and struggle they had to endure for giving their children the best of future. Words are not enough to thank them. Its a blessing to have them as parents. You will always inspire me to work hard and none of this is possible without your support and blessings. To my elder brothers Abhijeet and Arpit, you have been the two pillars of my life. Your love, encouragement and guidance regarding every thing related to life has what kept me going so far. To my lovely sister-in-law, Neeta, who fed me delicious food every day and took care of me selflessly. To Chizoku san, who helped me from the first day I came to Japan, for your motherly love and for being so kind and helpful to me. To my friends in Japan and back in India, specially Swapnil, Ronak, Sopan, Anurag, Ankur, and, Siraj. You have been my biggest supporter and I thank you all, for your friendship.

Lastly, I would like to thank my lovely wife, Sneha. None of this would have been possible without you. You are the best thing that happened to me. These 5 years had been challenging for us. Living away from each other was not easy but you supported me throughout my research studies despite the distances we shared. I thank you for your endless love and support, for standing by my side for all the tough decisions, through the thick and thin and above all, being my best friend. I thank you for everything.

Abstract

The work in this thesis addresses an important problem in mobile robot navigation called as the Simultaneous Localization and Mapping (SLAM) problem. Just like we humans perceive the world using our senses and make judgment as how to navigate in the real environment, robots need high level of intelligence and computation to make such judgment using onboard sensors. They also need to consistently update their belief about its position and uncertainty. SLAM is such a fascinating field that it gives any mobile robot intelligence to interact with the environment its operating in and make intelligent decisions on how to navigate inside such environments. To do so, they require an accurate map of the environment (2D or 3D), and autonomously navigate in structured environments. The applications of such kind of robots are many, mainly autonomous navigation, obstacle avoidance, as service robots, search and rescue operation, teleoperation and many more. This thesis aims to find solutions to construct accurate 2D and 3D maps of indoor environments for successful autonomous behavior of a wheeled mobile robot.

The thesis addresses three major solutions to the SLAM problem:

1. **Localization using state of the art filters:** Localization using existing solutions for estimation like the Extended Kalman Filter (EKF), particle filters, and graph theory are presented. The work briefly describes techniques for building maps using Laser Range Finders (LRF) for 2D mapping and localization. 2D features such as corner points and 2D lines are used as features for building robust maps. These are then incrementally built using the probabilistic methods. Since the robot is assumed to be starting with no map, the map is built as the robot moves around inside the environment.
2. **Clustering approach to build maps:** This work proposes many solutions to make accurate maps by utilizing clustering algorithms for effective noise removal, data segmentation, data extraction and building memory efficient maps for long time mapping of large environments. A novel line extraction methods using

clustering of LRF data is briefly presented which is computationally faster and robust to noise produced from cluttered areas.

3. **3D mapping using low cost RGBD sensors:** Finally, 3D mapping solution is presented that uses low cost depth or RGB-D sensors. Vision algorithms are used for fast feature extraction from depth images for constructing accurate maps providing rich information for safe navigation in cluttered areas. The image feature extraction method is combined with 2D localization method to make accurate 3D maps of large areas.

All the above algorithms are presented with results in real environment. The methods can easily be integrated to all kinds of wheeled robots for accurate navigation in indoor environments.

Thesis Outline

The thesis is divided into the following Chapters:

1. **Chapter 2:** This chapter gives a brief introduction to different probabilistic filter techniques that are used to address the SLAM problem. The main filtering algorithms like the Extended Kalman Filter, particle filter and Rao-Blackwellized particle filter along with graph based method are discussed in detail that forms the base for the rest of the chapters.
2. **Chapter 3:** This chapter introduces different feature extraction methods to extract and segment important features from the LRF data. The chapter gives a comprehensive details of extracting corners and line segments from the laser scans. The chapter also discusses how to generate the covariances of the end points and the corner points that are to be used for the EKF SLAM.
3. **Chapter 4:** This chapter forms a major work on a novel clustering techniques for feature extractions and line segment mapping from the LRF data. The major work is on extracting features from the LRF data from noisy environments. The noise from the cluttered environment needs to be removed prior to extracting important features. The contributions of the chapter are: (1) it shows the applicability of density-based clustering methods and mathematical morphological techniques generally used in image processing for noise removal from laser range sensor data; (2) it presents a new algorithm to generate straight-line maps by applying

clustering in the spatial domain; (3) it presents a new algorithm for robot mapping using clustering in a Hough domain; and (4) it presents a new framework to load, delete, install or update appropriate kernels in the robot remotely from the server. The framework provides a means to select the most appropriate kernel and fine-tune its parameters remotely from the server based on online feedback, which proves to be very efficient in dynamic environments with noisy conditions. The accuracy and performance of the techniques presented in this paper are discussed with conventional line segment-based EKF-SLAM and the results are compared.

4. **Chapter 5:** This chapter forms the formulation of SLAM algorithm using all the methods described in Chapter 2. The chapter presents the line segment based EKF algorithm for constructing line segment based maps with corner features. Next, the occupancy grid mapping approach using particle filter and Rao-Blackwellized particle filter is discussed with results to real environment. Results using standard scan matching and iterative closest point (ICP) technique are also discussed with results. The chapter also presents the solution to an important problem in SLAM called as the "Loop Closure" problem in which if the robot revisits a known area again in the environment after some time its hard to match it with its previous known belief and if not solved produces inconsistent maps. Graph based theory is used to solve the loop closure and results of long time large scale maps are presented.
5. **Chapter 6:** This chapter introduces 3D mapping solution to the SLAM problem using low cost RGBD sensors. Although, 2D maps can help robot navigate autonomously in large areas but in real environment 3D features makes more sense as the robot operates in a 3D world and not 2D. This chapter introduces some of the very popular vision algorithms like the Scale Invariant Feature Transpose (SIFT) and Speeded Up Robust Features (SURF) methods to extract 3D features from the depth sensors. A 3D ICP method is then used along with optimized graph theory to construct robust 3D maps of the environment.
6. **Chapter 7:** Chapter 7 concludes the thesis work with possible discussion on future works.

Table of Contents

List of Figures	xii
List of Tables	xvi
1 Introduction	1
1.1 Motivation	2
1.2 Thesis Outline	5
2 Theory and Background	7
2.1 Estimation	7
2.2 Environment Representation	7
2.3 Pose Estimation	8
2.3.1 Data Association	12
2.3.2 Dead Reckoning	14
2.3.3 Gaussian PDF	14
2.4 Belief presentation	14
2.4.1 Single-hypothesis belief	15
2.4.2 Multiple-hypothesis belief	15
2.5 Kalman Filter	17
2.5.1 Extended Kalman Filter	18
2.5.2 Data Association	20
2.6 Particle Filter	20
2.7 Rao-Blackwellized Particle Filter	22
2.8 Conclusion	24
3 Feature Extraction and Detection	25
3.1 Introduction	25
3.2 System Specification	25

3.2.1	Laser Range Finder Specifications	26
3.2.2	ROS: An open source operating system for robots	29
3.3	Data Detection	30
3.3.1	Split and Merge	31
3.3.2	Invariant Point Classification	33
3.4	Data Extraction	35
3.4.1	Line Segment Extraction	35
3.4.2	Corner Extraction	40
3.5	Conclusion	41
4	Clustering Approaches to Mapping	43
4.1	Introduction	43
4.2	LRF data clustering in spatial domain	43
4.2.1	Modes of LRF clustering	44
4.2.2	LRF Data Segmentation	47
4.3	Noise Removing Techniques	49
4.3.1	DBSCAN	50
4.3.2	Results- DBSCAN	54
4.3.3	Noise removal using Mathematical Morphology	57
4.4	Embarrassingly Parallel Noise Removal Technique from LRF data	59
4.4.1	Hopping Noise Removal Algorithm	60
4.4.2	Parallel implementation of the algorithm	65
4.4.3	Result	66
4.5	Clustering Algorithms	68
4.5.1	k-means	68
4.5.2	Fuzzy c-means Clustering	72
4.5.3	FCM Algorithm	72
4.5.4	Results - k-means and Fuzzy c-means	73
4.5.5	Generating Straight-line Maps	73
4.6	Clustering in the Hough Domain (Accumulator Space)	78
4.7	Framework	84
4.8	Experimental Results	86
4.8.1	Test Environment	87
4.8.2	Mapping Results	88
4.8.3	Comparison with Line Segment-based EKF-SLAM	98

4.9	Conclusion	100
5	SLAM- Simultaneous Localization and Mapping	101
5.1	Introduction	101
5.2	Robot Odometry Model	102
5.3	Kalman Filter localization - line feature extraction	105
5.4	Occupancy Grid Mapping (OGM)	115
5.5	Graph SLAM	118
5.6	Experiment Results	120
5.7	Conclusion	137
6	From 2D to 3D - Constructing Robust 3D Maps	138
6.1	Introduction	138
6.2	Microsoft Kinect	139
6.3	Feature Extraction	140
6.3.1	SIFT feature	141
6.3.2	SURF feature	143
6.3.3	Initial Matching	143
6.3.4	RANSAC	144
6.4	Feature Transform	147
6.4.1	3D Transformation Estimation	147
6.5	Pose Graph	149
6.5.1	Loop Closure	150
6.5.2	Pose Estimation	150
6.5.3	Optimization	154
6.6	Map construction	154
6.7	Experiment Results	155
6.8	Conclusion	156
7	Conclusion and Future Works	163
7.1	Future Works	163
	References	165
	Appendix A Robot Odometry Model	173
A.1	Robot Odometry	173

List of Figures

1.1	Autonomous robots in field	3
2.1	Example of pose distribution in one dimension. (a) Robot detects obstacle at distance. (b) PDF distribution around the robot.	9
2.2	Predict and measurement update cycle.	10
2.3	Different stages of the prediction and measurement update cycle in a one dimension case.	11
2.4	Belief update over different measurements.	13
2.5	Belief representation for mobile robot in 1D (continuous and discretized maps)	16
3.1	Pioneer P3DX	26
3.2	Robot specifications	27
3.3	Laser Range Finder specifications	27
3.4	clustering criteria for the data detection	30
3.5	Merge process for finding clusters	31
3.6	Split algorithm for finding linear segments.	32
3.7	Invariant point description	33
3.8	Invariant point classification	34
3.9	Line parameter in polar form	38
4.1	Cluster Analysis	44
4.2	LRF data segmentation	45
4.3	Modes of clustering	46
4.4	Geometry of a scanned point and a corresponding line.	48
4.5	Principle of IEPF algorithm	48
4.6	Core, border points and noise points in center-based density.	50
4.7	Description of border point and core point	52

4.8	Asymmetrical case for the DBSCAN algorithm.	52
4.9	Density reachability	53
4.10	Density connected points	53
4.11	Simple simulation environment.	54
4.12	DBSCAN result on sample LRF data.	55
4.13	DBSCAN result on the test environment	56
4.14	Structuring element kernel type.	57
4.15	Result of erode-dilate on LRF data.	57
4.16	Hopping window concept.	61
4.17	Reverse hopping criteria.	62
4.18	Parallel implementation of hopping window noise removing algorithm .	66
4.19	Results of noise reduction using Hopping window	67
4.20	Illustration of k-means algorithm.	70
4.21	Results of k-means and fuzzy c-means clustering on noise free data. . .	74
4.22	Straight line extraction from SVD, HT and centroids.	75
4.23	Line representation in ρ and θ parameter.	76
4.24	Map generated using clustering in the spatial domain.	79
4.25	Line generated after clustering the Hough Space parameters. The red line is the optimum line obtained by clustering in the Hough domain. .	80
4.26	Framework for robot map building showing the robot side and the server side. The message is exchanged in <code>xml</code> format between robot and server.	84
4.27	Error plot showing the percentage in error with the number of clusters using k-means and fuzzy c-means clustering.	87
4.28	The plots of the scan position error and odometry position error with respect to time.	88
4.29	Map generated using clustering in the spatial domain.	89
4.30	Map generated using clustering in the Hough domain.	90
4.31	Actual corridor test environment.	91
4.32	Raw laser scan of the real environment shown in Figure 4.31.	92
4.33	Result of the DBSCAN algorithm on (a).	93
4.34	Result of k-means clustering on (b) with clusters and centroids.	94
4.35	Line maps by joining the centroids of (c).	95
4.36	Result of the Hough transform on the centroid points of (c).	96
4.37	Line map obtained from clustering in the Hough domain. ($\delta = 0$) . . .	97
4.38	Time cost between EKF-SLAM and the proposed technique.	98

4.39	Map generated by line segment-based EKF-SLAM	99
5.1	Schematic flow of the Kalman filter localization	105
5.2	Kalman filter prediction step	107
5.3	LRF observations and line extraction	108
5.4	Target position representation in world and robot frame	109
5.5	Kalman filter matching step	110
5.6	Kalman filter estimation of the position step	111
5.7	SLAM process flowchart	113
5.8	Occupancy grid map example	117
5.9	Schematic representation of a graph and the resulting constraints . . .	118
5.10	Strong and Weak links in the graph nodes	119
5.11	EKF-line map of the corridor environment	121
5.12	Point map obtained from pure ICP	122
5.13	Estimated path from the robot with most likely path	123
5.14	Time taken by the ICP algorithm	124
5.15	Orientation of the robot over time for the corridor environment	124
5.16	Odometry over time for the corridor environment	125
5.17	Grid map from the particle filter algorithm	127
5.18	Comparison of the FastSLAM with number of particles.	128
5.19	Memory usage for particle size of 200.	128
5.20	Pose graph grid map of the corridor environment	129
5.21	Unoptimized pose constrains for the corridor environment	130
5.22	Optimized poses with the corrected loop closure constraints	131
5.23	Map obtained from pure odometry for the corridor environment	132
5.24	Loop closure failure for a large single loop	133
5.25	Loop closure corrected using offline optimization	134
5.26	Loop closure failure for long double loop map.	135
5.27	Loop closure corrected using offline optimization for the double loop. .	136
6.1	The Kinect sensor from Microsoft Corporation.	139
6.2	Difference of Gaussians at different scales and SIFT descriptor	142
6.3	SURF detector and approximation of Gaussian second derivative	143
6.4	SURF scale space built.. . . .	144
6.5	SURF keypoint description.	144

6.6	An example showing the matching of SURF features for a corridor environment	145
6.7	Wrong feature matching: Wrong feature detected during the matching and are ignored.	146
6.8	Illustration of RANSAC process. The green points are <i>Inliers</i> and orange points are the <i>Outliers</i>	147
6.9	Loop closure detection	151
6.10	Graph optimization process	152
6.11	3D map of assembled point cloud for Experiment 1.	157
6.12	Statistical analysis of Experiment 1	158
6.13	Trajectory before optimization, the failed loop closure and link constraint are shown in red.	159
6.14	Result of optimization of the graph nodes	159
6.15	3D map of assembled point cloud for a large loop.	160
6.16	Top view of the 3D map with colored point map.	161
6.17	Overlayed 3D map on actual building map, shows the accuracy of the map.	162
A.1	Robot coordinate frame system.	173

List of Tables

3.1	Pioneer P3DX Specifications	26
3.2	Hokuyo UHG-08LX Laser Sensor Specifications	28
3.3	Split and Merge parameters	34
3.4	Invariant point classification parameters	35
4.1	Definition of various kernels with parameters.	59
4.2	Table of Notations for k-means algorithm	69
4.3	Percentage error of the map area for clustering in the spatial domain for an artificial environment as shown in Figure 4.11.	86

Chapter 1

Introduction

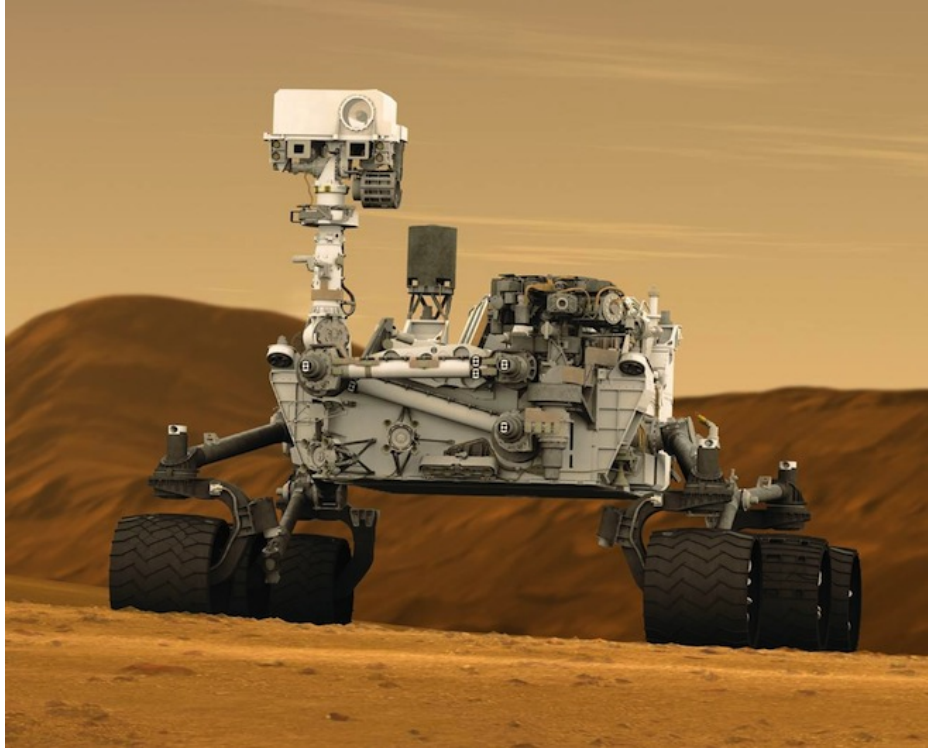
Robots have come a long way since the beginning of the industrial age. Traditionally, robots were meant to be bulky, with huge arms and fixed to a certain place away from human interaction. Early industrial robots brought a revolution into the manufacturing and production industries with rapid product development and round the clock use of robots in industries. At present, multiple robots are being employed in military, manufacturing, hospitals, disaster mitigation and for entertainment. The most significant change has been brought up by the advancement of artificial intelligence and machine learning techniques in robotics. This has made possible for robots to become completely autonomous in unstructured environments by utilizing sensors. Today there are robots operating in hazardous areas in applications such as search and rescue, mapping of collapsed mines, disaster site surveying, disposing dangerous chemicals and bombs and numerous other examples. Autonomous robots have become crucial for our quest to outerspace explorations. An example is the “Opportunity” rover by NASA, which is an autonomous robot exploring the unknown terrain at Mars[70]. Such is the impact of these autonomous bots that in 2005 a team led by Sebastian Thrun won the DARPA Grand Challenge, a prize competition for American autonomous vehicles, funded by the Defense Advanced Research Projects Agency, the most prominent research organization of the United States Department of Defense [103]. We already have driverless cars running on our streets that will take over manual drivers by year 2030. All these intelligent robots run on high complex algorithms and need accurate sensor data for navigation. A critical requirement of higher level navigation applications is that the robot has some reasonable knowledge of its current position with respect to a fixed reference frame. For example, in navigation applications that entail motion to a target position, the robot needs an accurate estimate of its

current position to plan a path to the goal and to confirm success. This thesis aims at such kinds of autonomous robots in indoor environments.

1.1 Motivation

There are three main problems in mobile robotics. The first problem can be described as a "*Localization*" problem or by a simple question of "*Where am I?*" in the world. This problem refers to the correct pose of the robot in the global coordinate frame of reference and is commonly termed as a "*Localization problem*". The process of position estimation with respect to a fixed reference frame is defined as the localization of the robot. The second problem for the mobile robot to autonomously operate in any structured or unstructured environment will be described by the question "*How do I go from point A to point B?*" and is commonly referred to as a "path planning problem". This will require the robot to accurately know its position or "localization" and to have some information of the environment to go from start to goal position. The third problem can be described in a simple question as to "*How does the world look like?*". This is referred to as the "*Mapping problem*" or in simple terms the correct information of the environment the robot is moving in by means of data collected by sensors. Clearly, all the three problems are critical and are very much co-related to each other because, when an accurate description (*map*) of the environment exists then the robot can "*localize*" itself by matching its observations using the sensors to the world reference model; also, if the robot knows its pose with high accuracy, then it can seamlessly navigate from point A to point B using the observed features, its pose and map. This process of trying to solve the above problems is termed as "*Simultaneous Localization and Mapping (SLAM)*" or "*Concurrent Localization and Mapping (CLM)*".

A mobile robot can localize itself using two different sets of sensors on board, proprioceptive sensors and exteroceptive sensors. Proprioceptive sensors, such as encoders or inertial measurement units (IMUs), measure the motion of the robot, acquiring data that can be integrated to estimate relative robot displacement. This method of localization is called odometry, or dead reckoning, and when used alone, the integrated error in global position grows without bound over time. Exteroceptive sensors, such as laser range scanners or cameras, take measurements from the external environment. This data can be correlated at subsequent robot positions to compute relative pose or displacement estimates, which can improve and sometimes replace odometry. Externally sensed data may also be correlated with data from a global map, giving a global position measurement and bounding the overall position error. If a



(a) The “Opportunity” rover on Mars by NASA.[70]



(b) Stanley autonomous car by the Stanford University that won the DARPA Grand Challenge.[103]

Fig. 1.1 Autonomous robots in field

global map is not initially available, it is possible for the robot to build a global map with the externally sensed data, while using this map to localize. This is the general concept of SLAM.

Most of the work in this thesis focuses on the localization and mapping using the exteroceptive sensors especially the Laser Range Finders (LRFs), which are planer horizontally fixed sensor giving range measurements of the obstacles. There have been many solutions to the SLAM problem proposed in the past decade. An earliest attempt to solve the SLAM problem was proposed in work given in [92], in which uncertainty models were introduced to solve the localization and mapping simultaneously. The trajectories and positions of landmarks in the environment are estimated without any need for prior knowledge of the location of the landmark. To get a complete perception of the region, the scan data from the sensors must be matched and fused in order to get a desirable map of the region [25]. The book by Thrun et.al [99] gives a good summary of different SLAM algorithms in context to probability theory. The work by Thrun using probabilistic approach gave a new forefront to solving the otherwise complex SLAM problem [69, 99].

There are two common techniques used in mobile robotics for mapping: (1) point-based matching, and (2) feature-based matching. In the first technique, measured data points are directly used for mapping, such as in [16] where they utilize range data to localize in a polygonal environment. Using an iterative least squares minimization technique, they proposed a matching algorithm between point images and target models in an *a priori* map. A study by [62] proposed a similar self-localization approach, but not necessarily in a polygonal environment. Their algorithm iteratively improves the alignment between two scans by minimizing the distance between them. A major drawback of using point-based matching is that it requires a large amount of memory to handle and the storage grows disproportionately for large environments, which in turn reduces performance. The other technique for working with scan data involves utilizing the raw measurement points into geometric features by transforming the raw scans into meaningful features. These features can then be used for matching and scan corrections. In comparison to point-based matching, feature-based matching techniques require less memory, are more efficient and provide more information about the environment. In addition, the extracted features, such as corners and line segments, can be subsequently used as building blocks for localization or mapping. Large data can be scaled down to simple geometric features, even for large environments. Because of these reasons, feature-based matching has been extensively researched and employed

into SLAM. Feature extraction, feature tracking and efficient mapping algorithms work with much less data and are more stable, specially when dealing with noise. In most scenarios in which mobile robots work (both indoors and outdoors), planar surfaces are common and they are typically modelled by line segments. Line segments are the simplest geometric primitives used to describe structural environments. The work in [71] presents an extensive survey and summary of line feature extraction algorithms from range sensors for indoor mapping, including split and merge [13], incremental [1], a Hough transform [19, 23, 34], line regression [4], RANSAC [32] and an expectation-maximization algorithm [74, 87]. Probabilistic occupancy grid mapping with particle filter is another method to store maps for robot navigation [72, 99].

1.2 Thesis Outline

The rest of the thesis is divided into the following Chapters:

1. **Chapter 2:** Chapter 2 gives a brief introduction to different probabilistic filter techniques that are used to address the SLAM problem. The main filtering algorithms like the Extended Kalman Filter [21], particle filter [69, 99] and Rao-Blackwellized particle filter [99] along with graph based method [39] are discussed in detail that forms the base for the rest of the chapters.
2. **Chapter 3:** Chapter 3 introduces different feature extraction methods to extract and segment important features from the LRF data. The chapter gives a comprehensive details of extracting corners and line segments from the laser scans. The chapter also discusses how to generate the covariances of the end points and the corner points that are to be used for the EKF SLAM.
3. **Chapter 4:** Chapter 4 forms a major work on a novel clustering techniques for feature extractions and line segment mapping from the LRF data. The major work is on extracting features from the LRF data from noisy environments. The noise from the cluttered environment needs to be removed prior to extracting important features. The contributions of the chapter are: (1) it shows the applicability of density-based clustering methods and mathematical morphological techniques generally used in image processing for noise removal from laser range sensor data; (2) it presents a new algorithm to generate straight-line maps by applying clustering in the spatial domain; (3) it presents a new algorithm for robot mapping using clustering in a Hough domain; and (4) it presents a new framework to

load, delete, install or update appropriate kernels in the robot remotely from the server. The framework provides a means to select the most appropriate kernel and fine-tune its parameters remotely from the server based on online feedback, which proves to be very efficient in dynamic environments with noisy conditions. The accuracy and performance of the techniques presented in this paper are discussed with conventional line segment-based EKF-SLAM and the results are compared.

4. **Chapter 5:** Chapter 5 forms the formulation of SLAM algorithm using all the methods described in Chapter 2. The chapter presents the line segment based EKF algorithm for constructing line segment based maps with corner features. Next, the occupancy grid mapping approach using particle filter and Rao-Blackwellized particle filter is discussed with results to real environment. Results using standard scan matching and iterative closest point (ICP) technique are also discussed with results. The chapter also presents the solution to an important problem in SLAM called as the "Loop Closure" problem in which if the robot revisits a known area again in the environment after some time its hard to match it with its previous known belief and if not solved produces inconsistent maps. Graph based theory is used to solve the loop closure and results of long time large scale maps are presented.
5. **Chapter 6:** Chapter 6 introduces 3D mapping solution to the SLAM problem using low cost RGBD sensors. Although, 2D maps can help robot navigate autonomously in large areas but in real environment 3D features makes more sense as the robot operates in a 3D world and not 2D. This chapter introduces some of the very popular vision algorithms like the Scale Invariant Feature Transpose(SIFT) and Speeded Up Robust Features (SURF) methods to extract 3D features from the depth sensors. A 3D ICP method is then used along with optimized graph theory to construct robust 3D maps of the environment.
6. **Chapter 7:** Chapter 7 concludes the thesis work and gives discussion on possible future works.

Chapter 2

Theory and Background

2.1 Estimation

"Estimation is the process by which we infer the value of a quantity of interest $\mathbf{x}$, by processing data that is in some way dependent on $\mathbf{x}$."

Uncertainty revolves around the estimation problem in robotics, else most of the robotics problems we encounter would have been solved simply using algebraic expressions. But when dealing with uncertainties in our measurements the problem becomes much harder and sophisticated. This part of the thesis will give a brief introduction of some of the probability techniques and theory to solve such problems.

2.2 Environment Representation

There can be many ways to represent an environment and have been stated in Chapter 1. Since this work is mainly focused in indoor environment, the following four types of maps are described below [50]:

1. **Topological map** The world is represented as a connected graph. The nodes in the graph corresponds to places of importance and edges are the connections such as corridors.
2. **Feature Maps** Geometric features from the environment are used to represent the world. Examples are corners of walls, doors, lines, and points features.
3. **Grid Maps** Instead of representing the maps to certain features, here the world is divided into a grid. Each cell of the grid presents small area/volume of the world and is given a certain value expressing the chance of that part of the environment being occupied or not.

4. **Appearance based methods** Instead of having an intermediate representation of sensor data in the form of grids or features, sensor data is directly used to form a function from sensor data to pose space.

2.3 Pose Estimation

In the Bayesian framework aims to find the estimate at a time given that we have the sensor measurements at all times.

$$p(\mathbf{x}_k | \mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_k) \quad (2.1)$$

where $\mathbf{x}_k$ is some time-varying signal and $\mathbf{z}_i$ are the measurements [50, 94]. This problem is called as the Bayesian filtering problem or stochastic filtering problem. When $p(\mathbf{x}_k | \mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_k)$ is well represented by a Gaussian, the mean and covariance give a full description of the state. Most localization methods represents a way of approximating the distribution $p(\mathbf{x}_k | \mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_k)$. Thus, estimating the pose is very much about representing uncertain pose information. Consider a simple one dimensional example in which a robot observes an obstacle at distance $\mathbf{x}_1$. If there is only one obstacle in the environment where the robot is operating after some time the probability distribution function (PDF) will look something similar to Figure 2.1 which shows an example of the pose distribution in case of 1D. Figure 2.1b shows the PDF distribution around the robot when it detects the obstacle at distance $\mathbf{x}_1$.

However, in real world the pose estimation is not one-dimensional but rather three-dimensional. Most of the sensors that are used for measurements are not perfect and have noise in the data. Also, there are many structures that are to be detected. Some of these structures are very hard to distinguish. The real challenge in robot localization lies in performing correct data association between the sensory data and the environment model. If the matching can be done, the localization problem is reduced to a standard estimation problem, where the sources of the all the measurements are known. The robot motion estimate can be easily be determined by using the information from the wheel encoders. If the size of the wheel and distance between them is known, the robot motion estimate can be found. The information given so by the encoders is called the odometric information. If the robot in the previous example continues to move forward its PDF will spread out towards right. From this example it is clear that the better the odometric information we have, the closer the prediction

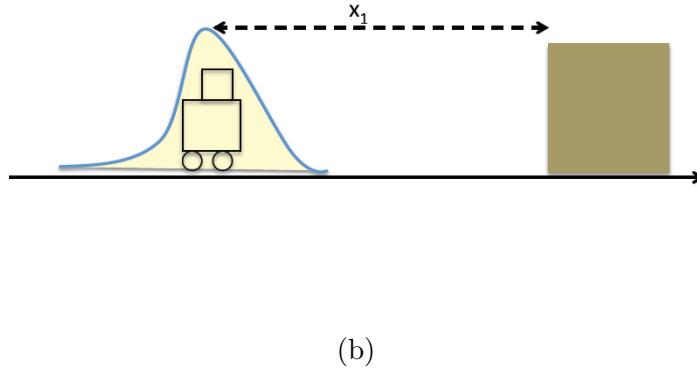
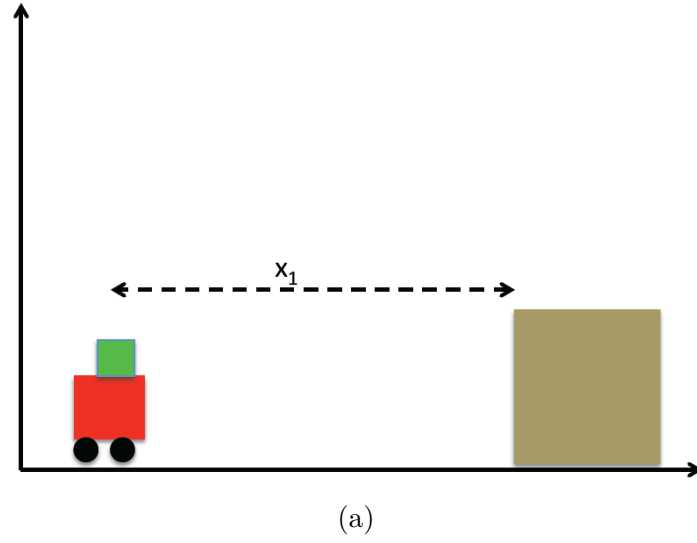


Fig. 2.1 Example of pose distribution in one dimension. (a) Robot detects obstacle at distance. (b) PDF distribution around the robot.

step is from the PDF. Hence, a good understanding of the odometry is necessary to make an accurate prediction and estimation.

The pose estimation problem can thus be divided into two steps, the *prediction* step and *measurement update* step which is given in Figure 2.2. In real world scenario, both the pose at any given time with correct measurement information is necessary. To understand the prediction and measurement update cycle problem let's consider a simple example of a robot in one dimension scenario as shown in Figure 2.3 for the case of localization. The measurement and update cycles can simply be termed as "see" and "act" for this example respectively. Consider a robot that has the map of the

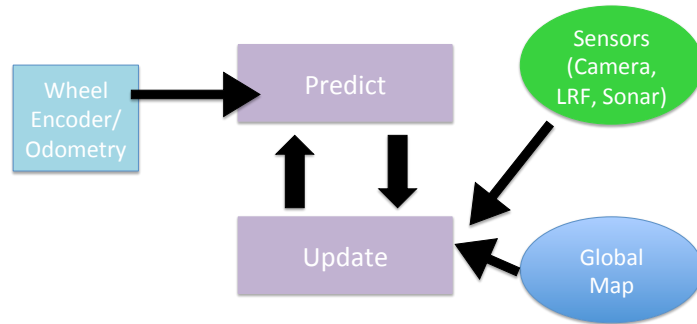


Fig. 2.2 Predict and measurement update cycle.

environment but has no clue where it is located in the environment. The environment consists of 3 similar looking pillars. In the beginning when the robot just wakes up, it has no prior information as it has taken no measurement so far, and so, the PDF is equally distributed over all of the environment as can be seen from Figure 2.3a. This means that it has the same knowledge of all possible positions in the environment.

Now, at the first point the robot takes the first measurement (from a laser sensor or camera) or “sees” the first object (a pillar) as shown in Figure 2.3b. After seeing the first pillar it can actually perceive a part of its environment. Now as the robot knows from the map there are 3 pillars, the robot can make the judgment to be in front of any of the three pillars. Hence, the probability distribution will change. As can be seen the robots perceive to be in front of any of the pillar with a Gaussian distribution that is equal for all the pillars as given in Figure 2.3c. In the next stage the robot moves around by action which moves the motors and moves the robot to the right as shown in Figure 2.3d. By moving around the robot actually collects the information about its wheel movement. Since this measurement is also uncertain because of the presence of slippage between the wheel and the ground or unequal wheel diameters. The probability distribution will change in this situation and if the robot continues to move forward without taking any new measurement its probability distribution will get worse and worse and it will have no clue as to where it is in the environment. This is shown by the blue distribution of the probability in Figure 2.3e.

In the next stage, the robot will again make the second measurement or see the next pillar in its way, as shown in Figure 2.3f. Again, this is the same situation as

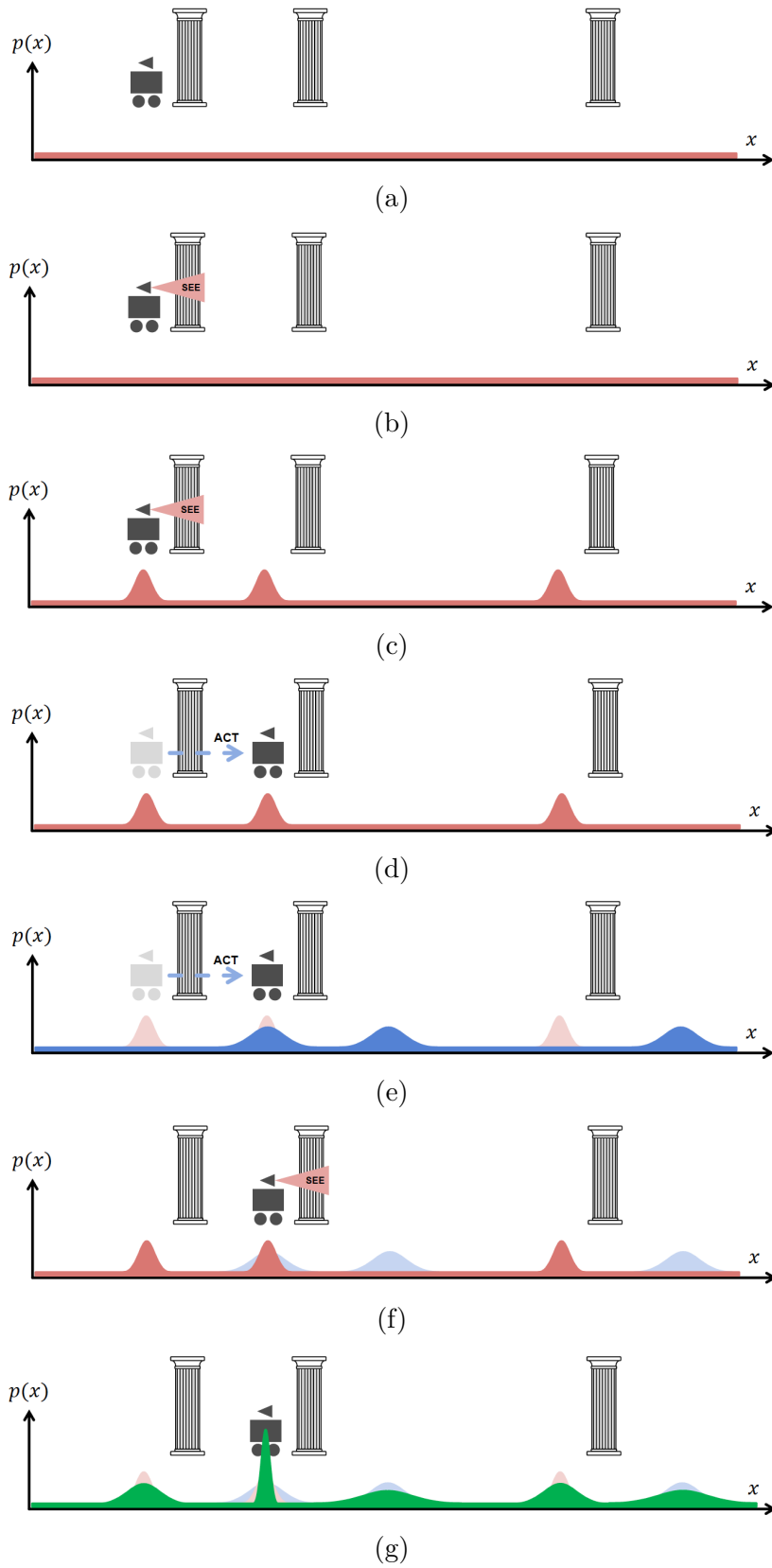


Fig. 2.3 Different stages of the prediction and measurement update cycle in a one dimension case.

the previous one as the robot sees only one pillar due to the limitation of its sensor's range. But, the robot now has a prior information from the previous movement which is shown in blue. Now, by fusing the new information coming from the perception with the previous estimate of the position, which is the blue distribution, the robot can infer a new situation, which is shown by the green distribution. The peak is the belief of the robot in its environment coming by fusing of two distributions that came from previous believes. And so, the robot if it can continue to do the cycle, will always know its position in the environment.

This is further explained in Figure 2.4. A robot is at position $\mathbf{x}_0$ as it starts and moves forward to positions $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, will have errors as it moves forward because of the reason discussed before (eg. wheel slippage) and its probability distribution will continue to grow and gets more uncertain shown in Figure 2.4a. The PDF spread out as it moves from $\mathbf{x}_0$ to $\mathbf{x}_3$. In Figure 2.4b, in the next stage, the robot takes an image of the environment and tries to perceive the environment. It then compares it with the map, and in doing this, the robot has a new estimate of the probability to make in a certain position and a certain observation. In this case, seeing this pillar shown in Figure 2.4c. Now, we have, in principle, two competing belief estimations. One is coming from the previous information. And another is coming from the "see" step. And now, if you fuse these two information's, which are both valid if you have the appropriate models, we can get a situation where we have much more precise knowledge about the robot's belief. We have a new position, which is the fusion of the two positions coming from two types of sensors (odometry and distance sensors). It's in this case, for example, a little bit closer to the perception x'_3 , because this has a lower uncertainty than the belief from before. And finally, the robot has a pretty good knowledge, and can use this knowledge to do, for example, further planning.

2.3.1 Data Association

An important problem in mobile robot localization and all sensor based estimation is that of the data association. The data association refers to the process of determining the origin of the measurements. In real world, when sensors take many measurements it is important to reject certain outliers as certain unmodeled objects in the environment cannot be used for localization. Either the map is augmented with the new structure or the measurement coming from it must be neglected.

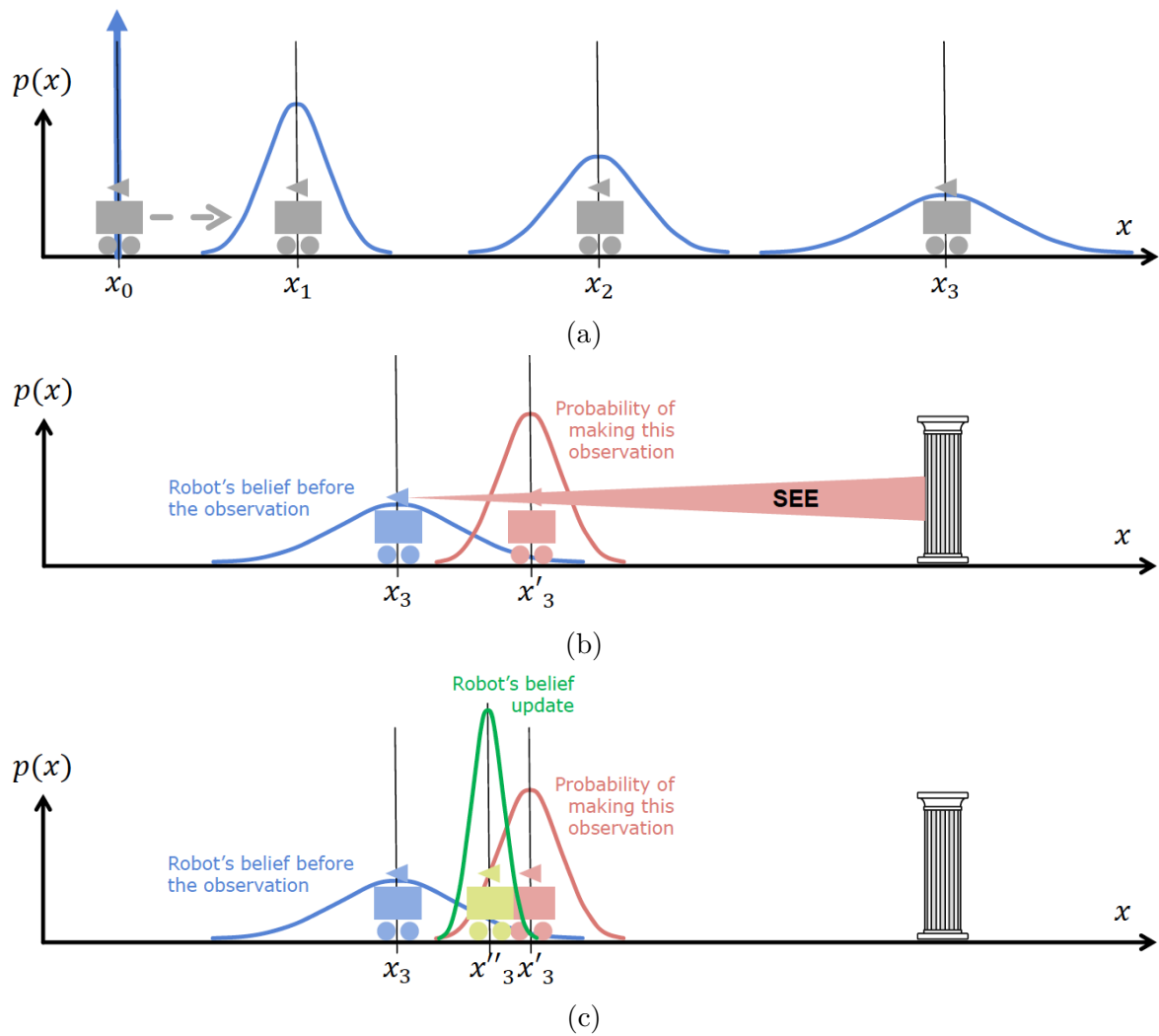


Fig. 2.4 Belief update over different measurements.

If $\mathbf{z}$ is the measurement and let $\mathcal{M}$ be the environment representation (or map). Data association can then be defined as for each $\mathbf{z}$ finding a correspondence in $\mathcal{M}$. With a feature based map the data association problem consists of classifying $\mathbf{z}$ as -

- a measurement of a map feature $f_i \in \mathcal{M}$, or
- a measurement of something not represented in the map.

Some of the methods for estimating the robot poses are discussed in the following section-

2.3.2 Dead Reckoning

If the robot is moved only for short distances, and the robot has good initial pose estimate, only the odometric information can be used to keep the track of robot poses. For long distances however, external sensors are needed to keep the pose uncertainties bounded.

2.3.3 Gaussian PDF

The Gaussian distribution is the most commonly used way to represent the uncertainty of a mobile robot. The reasons for using the Gaussian PDF are many. Firstly, because there are many different independent sources of uncertainty. Secondly, the knowledge about the uncertainty is seldom larger than knowing the first and second moment. A good thing about using Gaussian assumption filtering is that $p(\mathbf{x}_k | \mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_k)$ can be evaluated exactly. In one dimension the Gaussian function is completely specified by the mean $\bar{\mathbf{x}}$ and the variance $\sigma_{\mathbf{x}}^2$ by

$$\frac{1}{(\sigma_x \sqrt{2\pi})} e^{\frac{1}{2} \left(\frac{x - \bar{x}}{\sigma_x} \right)^2}. \quad (2.2)$$

When using higher dimensions, the value to estimate a vector $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ and the scalar variance becomes a covariance matrix $P = E(x - \bar{x})(x - \bar{x})^T$. The Gaussian PDF now becomes

$$\frac{1}{(|P| \sqrt{2\pi})} e^{\frac{1}{2} (x - \bar{x}) P^{-1} (x - \bar{x})^T}. \quad (2.3)$$

2.4 Belief presentation

The fundamental issue that differentiate various map-based localization systems is the issue of representation. There are two specific concepts that the robot must represent

and each one of it has its own specific solution. The robot must have representation of the environment or the map. What are the aspects of the environment that are present in the map?

The robot must also have representation of its belief regarding its position on the map. How does the robot identify unique position as its current position or possible positions? If there are multiple possible positions and they are expressed in a single belief, how to rank the multiple positions?

There are mainly two main belief representations that exist - single hypothesis and multiple hypothesis belief systems. In single hypothesis belief, the robot postulates to its unique position, whereas in the multiple hypothesis belief representation it is possible to describe the degree to which it is uncertain about its position.

2.4.1 Single-hypothesis belief

Single-hypothesis belief representation is the direct manipulation of the position of the robot in the map. Given a map, the robot's belief about the position is expressed as a single point in the map. Figure 2.5 (a) shows an example of single-hypothesis belief representation (Kalman filter). The advantage of the single-hypothesis representation of position is that given the unique belief, there is no position ambiguity. This means that the robot can simply assume that its belief is correct and can then select its future actions based on the unique position. Just as decision-making is facilitated by a single-position hypothesis, so updating the robot's belief regarding position is also facilitated, since the single position must be updated by definition to a new, single position. The challenge with this position update approach, which ultimately is the principal disadvantage of single-hypothesis representation, is that robot motion often induces uncertainty due to effector and sensor noise. Therefore, forcing the position update process to always generate a single hypothesis of position is challenging and, often, impossible.

2.4.2 Multiple-hypothesis belief

In the case of multiple-hypothesis beliefs regarding position, the robot tracks not just a single possible position but a possibly infinite set of positions. For example, the robot position's can be described as convex polygons positioned in a 2D map of the environment. This corresponds to the possible robot positions geometrically, with no preference ordering over the position. Each point in the map is in the robot's belief set and the one's outside the map are excluded. Such polygon representation of the

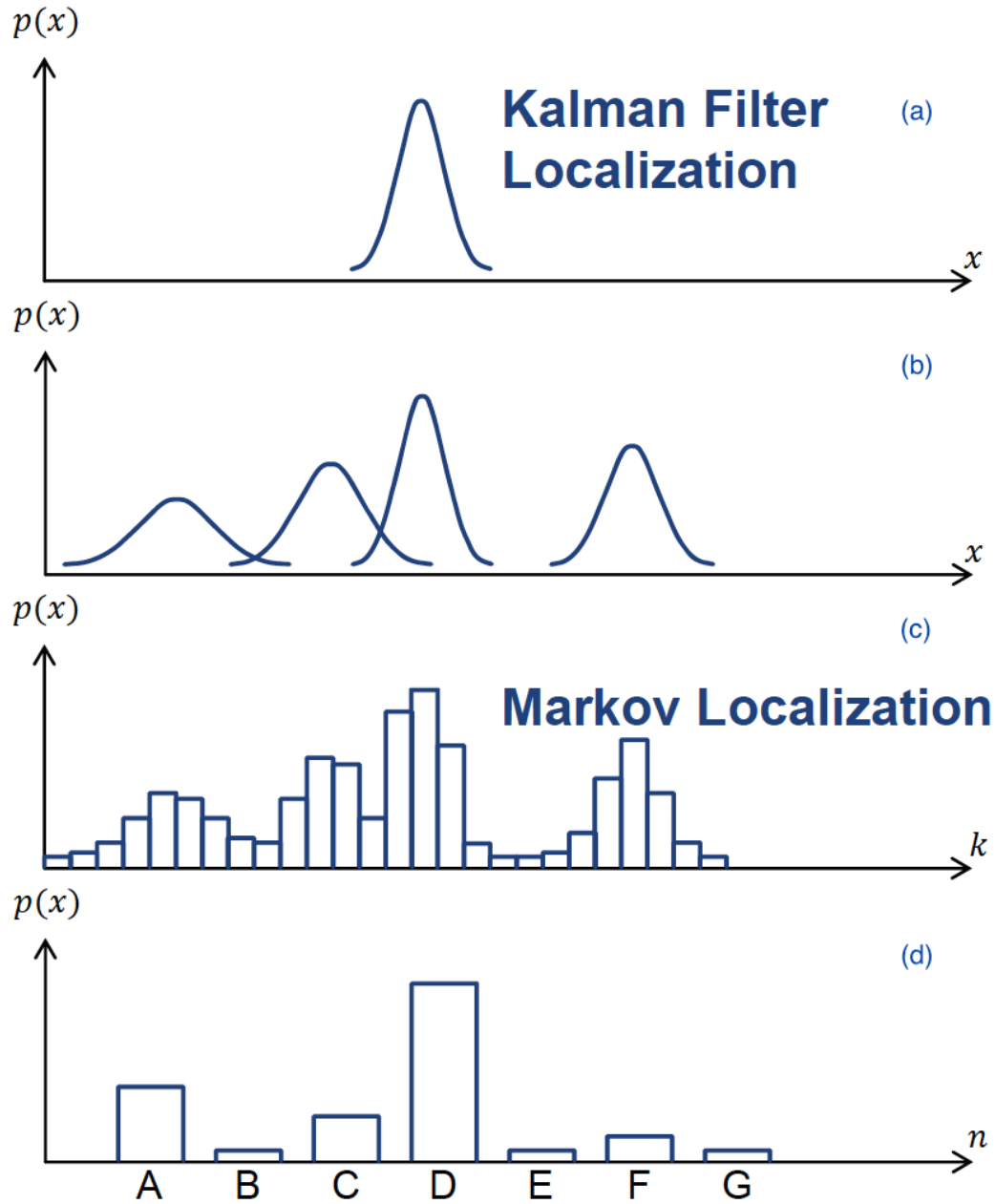


Fig. 2.5 (a) Continuous maps with single hypothesis belief (b) Continuous map with multiple-hypothesis belief (c) Discretized grid map with probability values for all possible robot positions (Markov approach) (d) Discretized topological map with probability value for all possible nodes (Markov approach).

multiple-hypothesis belief can apply to a continuous, geometric map of the environment [3, 57, 91]. The key advantage of the multiple-hypothesis representation is that the robot can explicitly maintain uncertainty regarding its position. If the robot only acquires partial information regarding position from its sensors and effectors, that information can conceptually be incorporated in an updated belief. In Figure 2.5 (b), shows an example of the multiple-hypothesis belief which are Gaussian in nature and represents different positions in the map. Figure 2.5(c) on the other hand, shows the grid representation or the Markov localization model where the position of the robot in the map is represented by grid with a varying probability distribution of the belief. Figure 2.5 (d), presents a similar Markov model but with nodes (A to G) with each node having a belief distribution of the position of the robot in the map.

2.5 Kalman Filter

The Kalman filter is a key component in many of the implementations for metric map representations (feature based), providing pose prediction, sensor fusion and data association. The first description Kalman filter was described in 1960's and has been extensively used for many applications [52] and is described in many books, e.g. [2, 37, 68]. The use of Kalman filter for mobile robot localization is given in [17]. A good introduction to Kalman filter can be found in [51, 102] .

Kalman filter is simply an optimal recursive data processing algorithm [68]. The key is that it is a recursive algorithm, and that the measurements can be incorporated into the estimate as they arrive and no batch processing is required. Since all the measurement data needs to be processed at each time step, the process soon becomes time consuming. Three assumption are necessary for the Kalman filter to provide the optimal estimate of the quantity $\mathbf{x}$ given some measurement $\mathbf{z}$. The system must be *linear* and the noise associated with the process model and measurement must be *white* and *Gaussian*. White noise does not exist in reality since it is not correlated in time and has the same frequency.

The Kalman filter algorithm can be divided into two parts, a *prediction* or time update step and a *correction* or measurement update step. Let $\mathbf{x}_k$ denote the true pose at the time step k . Let $\mathbf{u}_k$ be the input signal to the system, which could be control signal to the motors or odometric data. Here $\mathbf{x}_k$ is referred to as the state of the system and captures everything there is to know about the robot pose. This is the key to the recursive algorithm, everything that previous measurements have provided about the pose is encoded in $\mathbf{x}_k$. Assuming that the system is linear it can be written

as

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + G\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \quad (2.4)$$

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k \quad (2.5)$$

where F is referred to as the system matrix. The matrix G specifies how the input affects the state (pose), H is the measurement matrix, $\mathbf{w}_k \sim N(0, Q_k)$ is the process noise and $\mathbf{v}_k \sim (0, R_k)$ is the measurement noise. The process $\mathbf{w}_k$ and $\mathbf{v}_k$ are assumed to be independent. The matrix H encodes the map, i.e. the relation between robot pose and measurements.

2.5.1 Extended Kalman Filter

In a perfect world where F and G are known without error and with no process noise, can be used together with the input signal $\mathbf{u}_k$ to keep track of the robot pose if the initial pose is known. This is basically the assumption made when the odometry is used without any external measurements, in dead reckoning. In the presence of the noise and with an imperfect model there are errors. Let $\hat{\mathbf{x}}_{k|k-1}$ denote the *a priori* estimate of the robot pose at time step k using all measurements until the time $k-1$. Let also $\hat{\mathbf{x}}_{k|k}$ denote the *a posteriori* estimate of the pose when all measurements until time k are used. To keep track of the quality of the measurements, $\mathbf{P}_{k|k-1}$ the *a posteriori* estimation error covariance and $\mathbf{P}_{k|k}$ the *a posteriori* ditto. The prediction or time update step of the Kalman filter is given by

$$\hat{\mathbf{x}}_{k|k-1} = F\hat{\mathbf{x}}_{k-1|k-1} + G\mathbf{u}_{k-1} \quad (2.6)$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_{k-1}. \quad (2.7)$$

The measurements $\mathbf{z}_k$ taken from the system are used to correct the estimate. The innovation $\nu_k = \mathbf{z}_k - H\hat{\mathbf{x}}_{k|k-1}$ provides information that Kalman filter is unable to predict and is small when the prediction is good.

$$S_k = HP_{k|k-1}H^T + R_k \quad (2.8)$$

$$K_k = P_{k|k-1}H^T S_k^{-1} \quad (2.9)$$

$$\nu_k = \mathbf{z}_k - H\hat{\mathbf{x}}_{k|k-1} \quad (2.10)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k\nu_k \quad (2.11)$$

$$P_{k|k} = (I - K_k H)P_{k|k-1} \quad (2.12)$$

where K_k is the Kalman gain and S_k is the covariance of the innovation process. In the real world, many of the process are non-linear. The extension to the non-linear case is the extended Kalman filter (EKF), and is nothing but a linearization of the system around the current state estimate. The success of EKF depends on how well the system is approximated by the linearization. The non linear systems considered can be written as

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \mathbf{w}_{k-1}) \quad (2.13)$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k). \quad (2.14)$$

To get *a priori* estimate error covariance, the Jacobian of the function $\mathbf{f}$ has to be calculated with respect to $\mathbf{x}$ and $\mathbf{w}$. Defining

$$F_k = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}_k} \right|_{\hat{\mathbf{x}}_{k-1|k}, \mathbf{u}_{k-1}, 0} \quad (2.15)$$

$$W_k = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{w}_k} \right|_{\hat{\mathbf{x}}_{k-1|k}, \mathbf{u}_{k-1}, 0}, \quad (2.16)$$

the time update equation of the EKF are given by

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}, 0) \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + W_k Q_{k-1} W_k^T \end{aligned} \quad (2.17)$$

For the measurement update the Jacobians of the function $\mathbf{h}$ with respect to $\mathbf{x}$ and $\mathbf{w}$ are needed,

$$H_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{x}_k} \right|_{\hat{\mathbf{x}}_{k|k-1}, 0} \quad (2.18)$$

$$V_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{v}_k} \right|_{\hat{\mathbf{x}}_{k|k-1}, 0}. \quad (2.19)$$

The measurement update equations are given by

$$\begin{aligned}
 S_k &= H_k P_{k|k-1} H_k^T + V_k R_k V_k^T \\
 K_k &= P_{k|k-1} H_k^T S_k^{-1} \\
 \nu_k &= \mathbf{z}_k - \mathbf{h} \hat{\mathbf{x}}_{k|k-1,0} \\
 \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + K_k \nu_k \\
 P_{k|k} &= (I - K_k H_k) P_{k|k-1}
 \end{aligned} \tag{2.20}$$

The complete derivation of the Kalman filter equations can be studied in detail from [99].

2.5.2 Data Association

For the standard estimation techniques, like that of the Kalman filter, it is assumed that the correspondence between the measurement and the map is known. It is only by knowing this that the measurement equation can be defined properly. One common way of handling data association is to use the *Mahalanobis distance*, which is the Euclidean distance normalized with the covariance. The Mahalanobis distance ρ is defined by

$$\rho_k = \nu_k^T S_k^{-1} \nu_k. \tag{2.21}$$

The Mahalanobis distance defines a χ^2 distribution [6]. To accept a match between a measurement and its believed origin in the environment the Mahalanobis distance has to fulfill

$$\rho_k \leq \gamma \tag{2.22}$$

where γ determines the size of the validation gate. The value of γ depends on the dimension of the measurement and with that probability a true match should be classified as such. The size of the validation gate increases with $P_{k|k-1}$ and R_k as defined earlier in EKF equations.

2.6 Particle Filter

The particle filter approximates the optimal Bayesian filter by representing the robot position through arbitrary multimodal probability distribution using a set of particles [38, 86, 99].

$$\mathbf{X}_k = \{x_k^{[1]}, x_k^{[2]}, \dots, x_k^{[M]}\}, \tag{2.23}$$

Algorithm 1: Particle Filter algorithm

```

1  $X_k = \text{filter}(X_{k-1}, u_k, z_k);$ 
2 for  $m = 1$  to  $M$  do
3   | sample  $x_k^{[m]} \sim P(x_k|u_k, x_{k-1});$ 
4   |  $w_k^{[m]} = P(z_k|x_k^{[m]});$ 
5   |  $\hat{X}_k = \hat{X}_k + \langle x_k^{[m]}, w_k^{[m]} \rangle$ 
6 end
7 for  $m = 1$  to  $M$  do
8   | draw  $i$  with probability  $\propto w_k^{[m]}$  from  $\hat{X}_k$  ;
9   | add  $x_k^{[i]}$  to  $X_k$ 
10 end
11 return  $X_k$ 

```

Where M is the number of particles. For a single state or pose of the robot (one particle), $x_k^{[i]}$ is associated with importance factor or weight :

$$\langle x_k^{[i]}, w_k^{[i]} \rangle = P(x_k^{[i]} | z_{1:k}, u_{1:k}), \quad (2.24)$$

Here, the weights are the probability of the particle and is continuously updated after new observations are made:

$$P(x_k | z_{1:k}, u_{1:k}) = \eta \cdot P(z_k | x_k) \cdot P(x_k | z_{1:k-1}, u_{1:k}). \quad (2.25)$$

From Equation (2.25), the a priori estimate $P(x_k | z_{1:k-1}, u_{1:k})$ is computed by

$$P(x_k | z_{1:k-1}, u_{1:k}) = \int P(x_k | x_{k-1}, z_{1:k-1}, u_{1:k}) \cdot P(x_{k-1} | z_{1:k-1}, u_{1:k}) dx_{k-1} \quad (2.26)$$

$$= \int P(x_k | x_{k-1}, u_k) \cdot P(x_{k-1} | z_{1:k-1}, u_{1:k}) dx_{k-1}, \quad (2.27)$$

The idea behind particle filter is that Equation (2.25) can be approximated using set of particles given by Equation (2.23), where a single state is

$$x_k^{[m]} \approx P(x_k | z_{1:k}, u_{1:k}). \quad (2.28)$$

Algorithm 1 describes the pseudo code for the particle filter algorithm [86]. The temporary variable $\hat{X}_k$ contains the list of particles and their likelihood which are denoted by $x_k^{[m]}$ based on the previous particle $x_{k-1}^{[m]}$ and for the control input u_k . The probability of the sampled hypothesis is proportional to the observation probability

$P(z_k|x_k^{[m]})$. The observation probability is based on the difference between the current observation and the predicted observation according to the stored map of the sampled particle $x_k^{[m]}$. The resampling step is given by lines five to seven. In resampling step, a set of new particle set X_k are created out of the current $\hat{X}_k$ to reduce the variance of the underlying distribution. Particles with a higher weight will appear more often in the new list than ones with lower likelihoods. This means that the good hypothesis of the robot poses will remain in the non-parametric representation of the state and others will vanish. Different resampling strategies have been proposed in the book by Thrun et.al [22, 99].

Recently, particle filters have been extensively used for different filtering, navigation and tracking applications for mobile robots [7, 14, 104]. However, the performance of particle filter is lower when compared with the traditional Kalman filter approach. This is due to high computational cost for the resampling step, as this step creates copies of the particles for re-use. Methods such like *Gmapping*[40] and DP-SLAM[27] resample only when the particle weight variance is above threshold. The particle weight variance is given by:

$$N_{eff} = \frac{1}{\sum_{m=1}^M [(w_k^{[m]})^2]} \quad (2.29)$$

Here, N_{eff} is maximal for equal weights and resampling will not reduce the variance of the represented probability distribution. An important might problem occurs when the resampling step replaces the good samples and the final particles loses track of the correct state when the M is too small.

2.7 Rao-Blackwellized Particle Filter

The Rao-Blackwellized particle filter can be seen as a combination of Particle Filter and Extended Kalman Filter.

Usually, the map consists of features (corners, line segments) of the environment. The particle filter models the robot pose, and each map feature is represented by an Extended Kalman Filter. This gives an efficient solution to the SLAM problem which often referred to as *FastSLAM* [99]. The Kalman filter updates the full system Jacobians (Equation (2.14)), while an update here only affects one feature. That is why the particle filter scales exponentially while, the Kalman filter scales between linearly and quadratically with the number of dimensions of the estimation problem. Each particle represents one hypothesis of the robot pose and contains its own set of map features of the map. The map feature are denoted by μ and certainty is denoted by Σ .

Algorithm 2: Rao-Blackwellized Particle Filter algorithm

```

1  $X_k = \text{filter}(X_{k-1}, u_k, z_k, c_k);$ 
2  $\hat{X}_k = X_k = 0;$ 
3 for  $m = 1$  to  $M$  do
4   sample  $x_k^{[m]} \sim P(x_k | u_k, x_{k-1})$   $j = c_k;$ 
5   if feature  $j$  never seen before then
6      $\mu_{j,k}^{[m]} = h^{-1}(z_k, x_k^{[m]});$ 
7      $H = h^{-1'}(x_k^{[m]}, \mu_{j,k}^{[m]});$ 
8      $\Sigma_{j,k}^{[m]} = H Q_k H^T$ 
9   else
10     $\tilde{z} = h(\mu_{j,k-1}^{[m]}, x_k^{[m]});$ 
11     $H = h'(x_k^{[m]}, \mu_{j,k}^{[m]});$ 
12     $Q = H \Sigma_{j,k-1}^{[m]} H^T + Q_k;$ 
13     $K = \Sigma_{j,k-1}^{[m]} H^T Q^{-1};$ 
14     $\mu_{j,k}^{[m]} = \mu_{j,k-1}^{[m]} + K(z_k - \tilde{z});$ 
15     $\Sigma_{j,k}^{[m]} = (I - KH) \Sigma_{j,k-1}^{[m]};$ 
16     $w^{[m]} = \frac{1}{\sqrt{|2\pi Q|}} \exp \left\{ -\frac{1}{2} (z_k - \tilde{z})^T Q^{-1} (z_k - \tilde{z}) \right\}$ 
17   end
18    $\bar{X}_k = \bar{X}_k + \langle x_k^{[m]}, w_k^{[m]} \rangle$ 
19 end
20 for  $m = 1$  to  $M$  do
21   draw  $i$  with probability  $\propto w_k^{[m]}$  from  $\bar{X}_k$  ;
22   add  $x_k^{[i]}$  to  $X_k$ 
23 end
24 return  $X_k$ 

```

The mean is labelled with $\mu_{n,k}$

$$x_k^{[m]} = < [(x, y, \theta)^T]^{[i]}, (\mu_{1,k}^{[i]}, \Sigma_{1,k}^{[i]}), (\mu_{2,k}^{[i]}, \Sigma_{2,k}^{[i]}), \dots, (\mu_{N,k}^{[i]}, \Sigma_{N,k}^{[i]}) >, \quad (2.30)$$

where, $(x, y, \theta)^T$ is one robot hypothesis, i is the particle index and k denotes the point in time. The list of μ and Σ denotes the map. Algorithm 2 denotes the Rao-Blackwellized Particle filter which is very similar to particle filter algorithm (Algorithm 1).

The correspondence variable is denoted by c_k and is computed by the data association method which correlate observations with the stored map. Details about the data association technique can be found in the book by Thrun et. al [99]. The fourth line shows the generation of hypothetical state $x_m^{[m]}$ based on the previous state $x_{k-1}^{[m]}$ and control input u_k . Lines ten to sixteen is nothing but the Extended Kalman Filter update step for each feature as described before.

For the new features, only parameters $\mu_{j,l}^{[m]}$ and $\Sigma_{j,k}^{[m]}$, representing the map feature are initialized (lines six to eight). Function h^{-1} denotes the transformation from local to global coordinate system and H is the Jacobian of h^{-1} with respect to the observed feature z_k . Vector $\tilde{z}$ denotes the measurement prediction and is used to update the mean $\mu_{j,k}^{[m]}$ (line fourteen) and to calculate the importance factor $w^{[m]}$ (line sixteen). Transformation from local to global coordinate system is done by h and H is the corresponding Jacobian with respect to global map feature $\mu_{j,k}^{[m]}$.

Resampling is given in line twenty to twenty-four, which is similar to the particle filter algorithm. The complete derivation of the Rao-Blackwellized particle filter algorithm can be found in the book by Thrun et. al [99].

2.8 Conclusion

This chapter briefly presented some of probabilistic filters for estimation. Most of the filters are build upon Bayes theory and uses Gaussian filters. One of the primary advantages of Gaussian filters is computational. The update requires time polynomial in the dimensionality of the state space. The cases for unimodal or multi-model estimation are described briefly. The advantages and disadvantages of which filter to use depends on the resources, sensor and type of environment. The EKF approach is more robust but computationally expensive when the size of the environment is large. On the other hand, particle filters seems to be more robust to size and is more popular. In the subsequent chapters, a SLAM system is built using the presented filters.

Chapter 3

Feature Extraction and Detection

3.1 Introduction

In the previous chapters different probabilistic filtering techniques for localization and mapping were introduced. For a mobile robot different sensors can be used to get the information of the environment. Common sensors that are utilized in mobile robotics are the ultrasonic sensors, vision sensors and laser sensors. Once the data has been obtained, it is important to extract landmarks or features from the data for localization and for the purpose of building map of environment using those features. Common features that are utilized in the mobile robotics studies in indoor environment are corner features, line segments, color blobs, doors, windows etc. This chapter introduces some of the techniques that were used for the thesis work to extract meaningful features in form of corners and line segment that are later fused into the EKF framework for mapping. As localization is performed in a controlled scenario, without dynamic objects or people moving in the navigation area horizontal laser scan points are utilized in the data detection and extraction mode, so as to obtain linear segments and invariant points which are used for the localization.

3.2 System Specification

Figure 3.2 shows the robot with the sensors that was used during all the experiments. The base robot is the Pioneer P3DX from the Adept Mobilerobots®. Pioneer 3-DX is a small lightweight two-wheel two-motor differential drive robot ideal for indoor laboratory or classroom use. The robot comes complete with front SONAR, one battery, wheel encoders, a microcontroller with ARCOS firmware, and the Pioneer SDK advanced mobile robotics software development package. Pioneer research robots

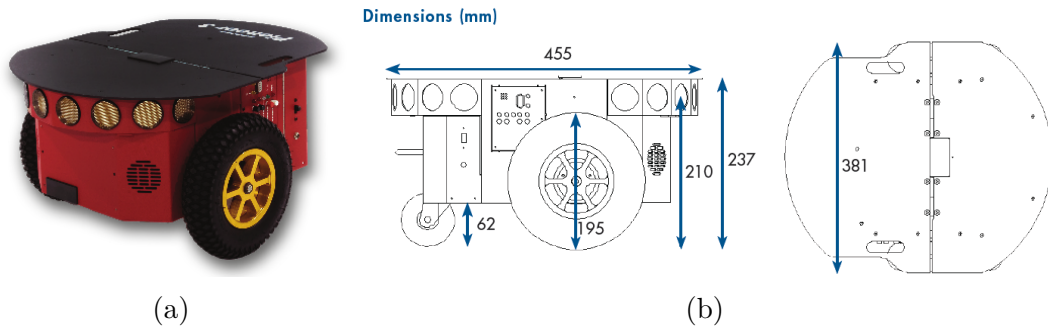


Fig. 3.1 Pioneer P3DX. (a) Differential drive base. (b) Dimensions of the robot.

are the world's most popular intelligent mobile robots for education and research. Their versatility, reliability and durability have made them the preferred platform for advanced intelligent robotics. Pioneers are pre-assembled, customizable, upgradeable, and rugged enough for extended period of use. Figure 3.1 shows the differential base with dimensions of the robot. Important specifications of the Pioneer P3DX is given in Table 3.1.

Table 3.1 Pioneer P3DX Specifications

Model	Pioneer P3DX
Traversable Terrain	Indoor
Robot Weight	9 kg
Operating Payload	17 kg
Turn radius	0 cm
Swing radius	26.7 cm
Rotation speed	300°/s
Max. Forward/backward speed	1.2 m/s
Max. Traversable Step	2.5 cm
Max. Traversable Grade	25%
Run time	8-10 hrs with 3 batteries
Voltage	12 V
Available Power Supply	5 V, 1.5 A and 12 V, 2.5 A

3.2.1 Laser Range Finder Specifications

UHG-08LX Scanning Laser Ranger finder shown in Figure 3.3 is used throughout the experiment and it is a range finder typed laser scanner. It is light weighted and fast scanning laser. The specialty of this range sensor is that it senses the 2 dimensional environments around it. The output is the distance to each object from the centre of the sensor. The features of the sensors are:

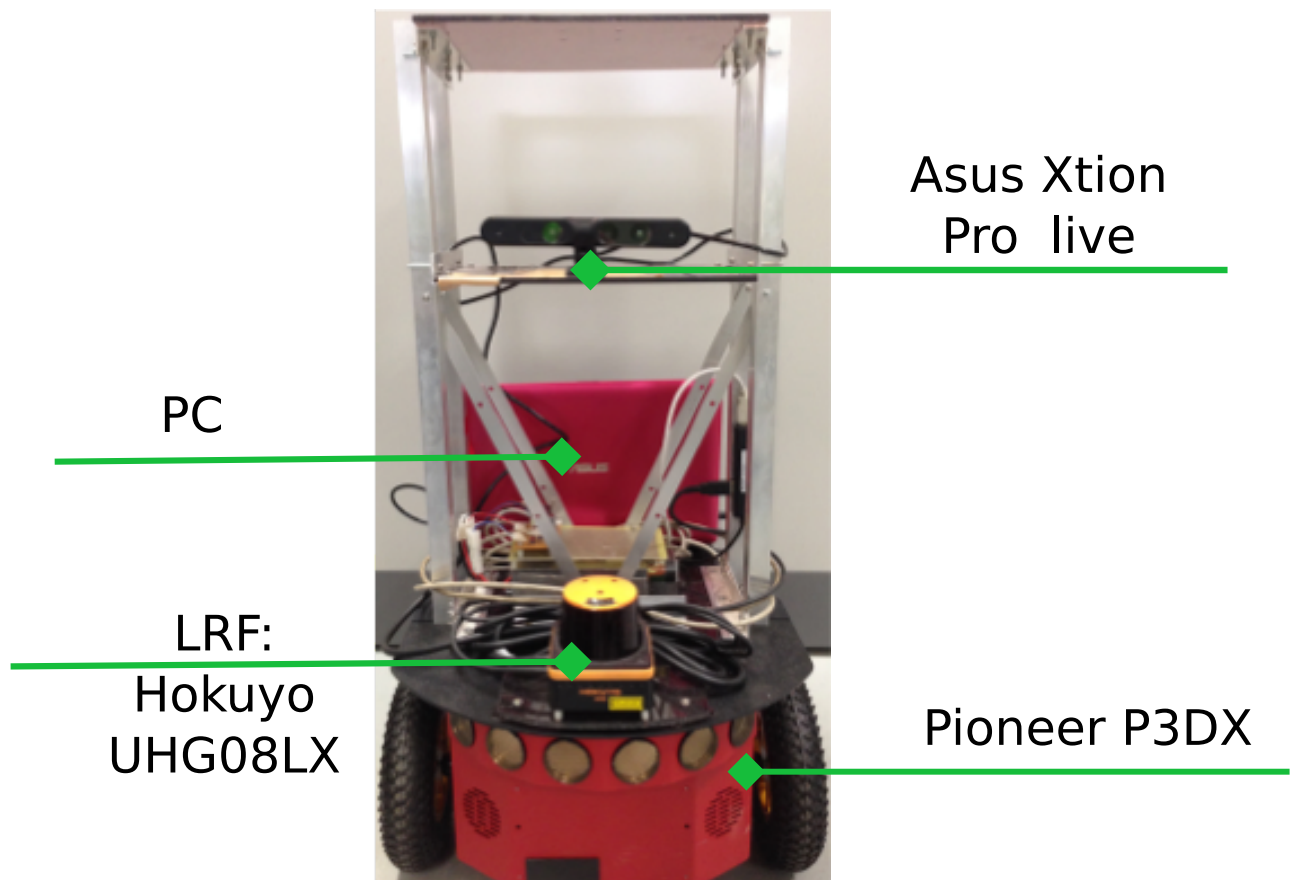


Fig. 3.2 Robot Specifications: Pioneer P3DX (differential drive base), UHG08LX (LRF), Asus Xtion pro live (Depth sensor) and PC

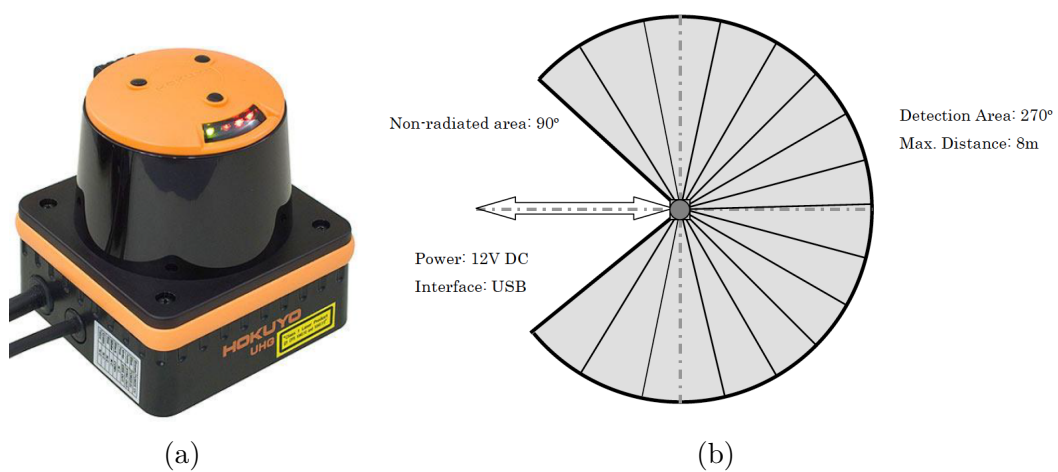


Fig. 3.3 Laser Range Finder specifications. (a) Hokuyo UHG08LX. (b) Range of measurement.

Table 3.2 Hokuyo UHG-08LX Laser Sensor Specifications

Type	UHG-08LX
Range Principle	Phase difference system due to the duration of flight of light
Illuminant	Semiconductor laser $\pm = 785$ nm, FDA laser safe class 1.
Laser Power	Below 0.8 mW
Power supply voltage	DC 12 V $\pm$ 10%
Electric Power Consumption	500 mA
Detection distance and detection body	20~ 8000 mm and white paper 250 mm
Range Precision	Under standard environment 100~1000 mm: $\pm$ 30 mm and 1000~8000 mm distance 3%
Range Resolution	1 mm unit
Scan Angular Range	270 degree
Angular Resolution	Approx. 0.36 degree, 1024 steps
Scanning time	67 ms/scan
Interface	RS-232C (19.2k, 57.6k, 115.2k, 500k and 750kbps) USB Ver2.0 FS mode (12 Mbps) OUTPUT 1 point (Synchronous o/p)
Mass	Approx. 500 g
External Size	(W)50 mm $\times$ (D)50 mm $\times$ (H)70 mm

- High accuracy, high resolution and wide angle, which provide the best solution for autonomous robots moving in the unknown environment.
- Compact size allows more designing freedom. Lightweight and low power consumption contributes to the long time operations.
- No influence by how bright the environment will be. Excellent performance in darkness.
- Recognizing human body's size and position without hurting their privacy.

however this sensor cannot be used outside, it is designed for indoor used only. An infrared laser with wavelength of 785 nm is used as a light source for the URG-04LX laser scanner. Scanning area is 270 degrees with 0.36 degrees/pitch (360/1024) as shown in Figure 3.3 and it can detect the distance and the direction to the objects. The detectable range varies from 0.02 to 8 m. UHG-08LX is suitable for identifying the environment around the sensor because it executes the distance data per each step as detected result. The measuring principle applied here is, phase difference system with time of flight. Therefore it is not influenced so much against the colour of objects or the surface gloss and make the stable measurements. Figure 3.3 (b) shows the scanning range of this sensor with detectable and non-detectable areas. The specifications of the UHG-08LX are given in Table 3.2.

3.2.2 ROS: An open source operating system for robots

For computation a dedicated PC was used on the robot for carrying out all the heavy computations for map building and navigation. The robot PC has a memory of 12 GB with an intel i5 processor. It has a storage capacity of 320 GB. The PC does not have a dedicated GPU (Graphical Processing Unit). For the programming, Python, C++, and Java was used. All computations were performed on the Linux system with Robot Operating System (ROS).

Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages [77].

Software in the ROS Ecosystem can be separated into three groups:

1. Language- and platform-independent tools used for building and distributing ROS-based software;
2. ROS client library implementations such as roscpp, rospy, and roslisp;
3. Packages containing application-related code which uses one or more ROS client libraries.

Both the language-independent tools and the main client libraries (C++, Python, and LISP) are released under the terms of the BSD license, and as such are open source software and free for both commercial and research use. The majority of other packages are licensed under a variety of open source licenses. These other packages implement commonly used functionality and applications such as hardware drivers, robot models, datatypes, planning, perception, simulation tools, and other algorithms.

3.3 Data Detection

In the following section some sets of important definitions will be defined for data detection.

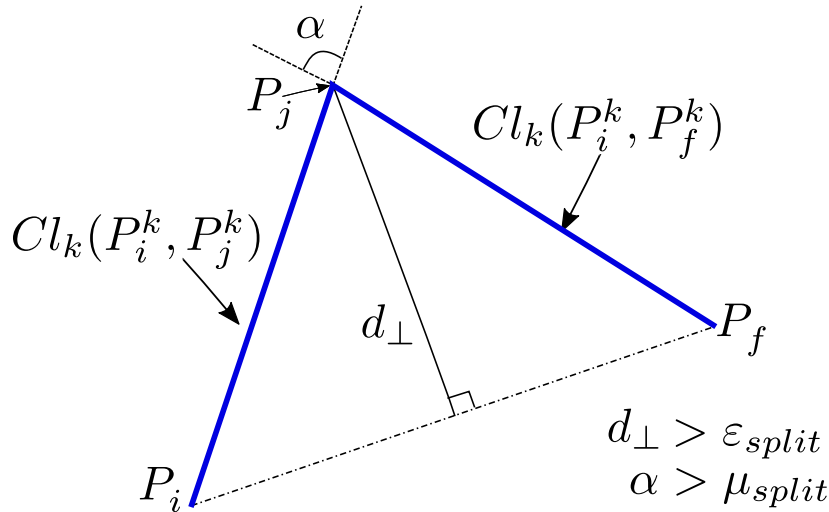


Fig. 3.4 Clustering criteria and measure of $d_{\perp}$ and α

1. **Definition 1** A list of points P_{2D} is represented by the vector $[P_1, \dots, P_j, \dots, P_n]$.

2. **Definition 2** From Figure 3.4, Cl_k is the list of connected points which are called the "clusters" with initial and final point as P_i^k and P_f^k .
3. **Definition 3** Lin_k is a set of connected points or the "line segments", with extreme points given by P_i^k and P_f^k .
4. **Definition 4** The value of $d_\perp$ is the perpendicular distance between $P_j^k \in Cl_k$ and the line segment connecting P_i^k to P_f^k .
5. **Definition 5** The value of α is the orientation between $Cl_k(P_i^k, P_j^k)$ and $Cl_k(P_j^k, P_f^k)$ as given in Figure 3.4, where $Cl_k(P_i^k, P_j^k)$ is the line segment that connects P_i^k and P_j^k , and the $Cl_k(P_j^k, P_f^k)$ is the line segment that connects P_j^k and P_f^k .
6. **Definition 6** The cluster Cl_k can be considered a candidate for line segment or set of linear segments. In Figure 3.5, an example of cluster representing candidates for linear segments is Cl_3 .

3.3.1 Split and Merge

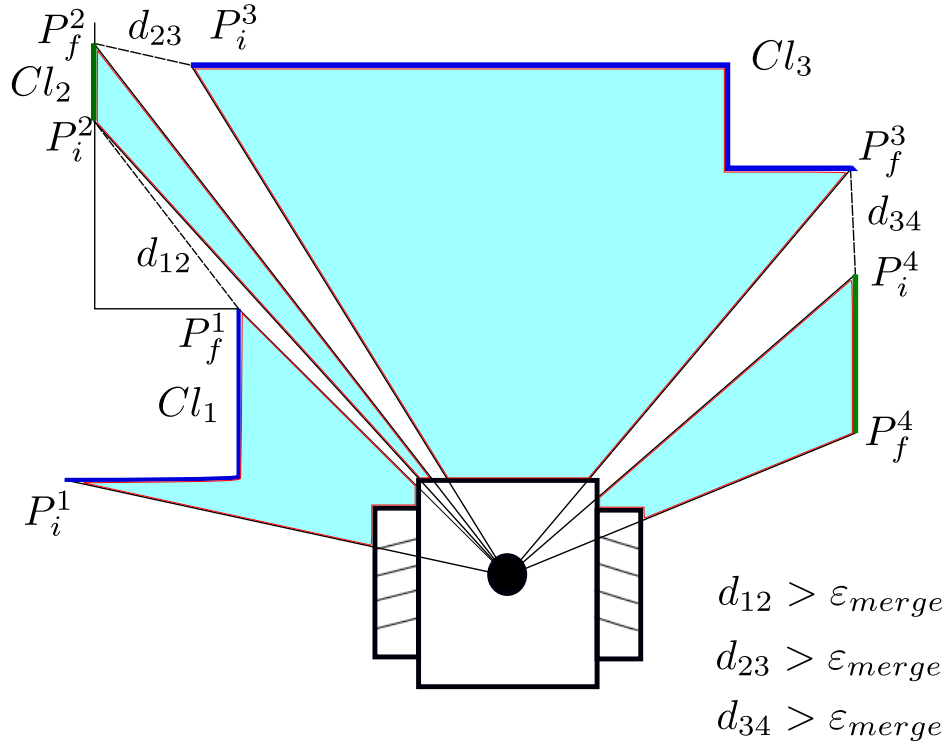


Fig. 3.5 Merge criteria and cluster $Cl_{P_{i,f}}$

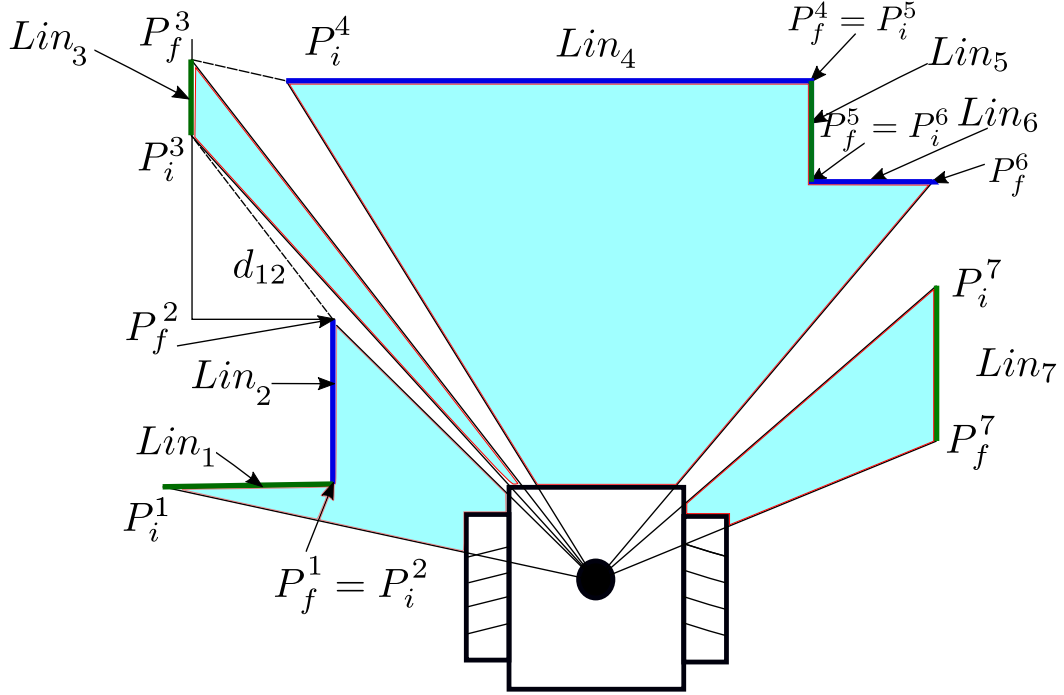


Fig. 3.6 Split algorithm for finding linear segments.

The method to detect the line segments from the set of LRF (Laser Range Finder) data is the Split and Merge algorithm [71]. The split and merge algorithm can effectively extract line segments from walls, doors and furniture in a static indoor environment and can also detect invariant points (corners, circular segments), at the same time. The flow of the algorithm is given below [75]-

1. **Merge:** The entire set of data points P_{2D} is iterated if points P_j and P_{j+1} have an Euclidean distance lower than ε_{merge} . When this happens, P_j and P_{j+1} are merged in the same cluster. The merge result generates a list of clusters Cl_k , shown in Figure 3.5. Next, to each cluster given by Cl_k , line segment candidates, split phase is applied. The line segment candidates are shown in Figure 3.5 with Cl_1 , Cl_2 and Cl_3 .
2. **Split:** Each cluster Cl_k which is classified as candidates for the line segment, the point $P_j^k \in Cl_k$ is searched, whose distance $d_{\perp}$ is the greatest. Then, Cl_k is iteratively split into clusters $Cl_k(P_i^k, P_j^k)$ and $Cl_k(P_j^k, P_f^k)$ which is given in Figure 3.4 with the following conditions
 - (a) $d_{\perp}$ is greater than ε_{split} .
 - (b) α is lower than μ_{split} .

When the iteration stops and the cluster Cl_k cease to divide, it is considered a linear segment Lin_j . Figure 3.6 shows the split phase.

3.3.2 Invariant Point Classification

After applying the merge and split algorithm, linear segments Lin_j are obtained with end points P_i^j and P_f^j . For the line segments Lin_j , the extreme points are tested to find whether they are invariant or not. For the line segment Lin_j and Lin_{j+1} with extreme points P_i^j and P_f^{j+1} , P_i^{j+1} and P_f^{j+1} respectively, the points P_f^j and P_i^{j+1} have distance equal to R_f^j and R_i^{j+1} , respectively. The angle distance between them is given by $\theta_{f,i}^{j,j+1}$ shown in Figure 3.7.

The test of invariancy for points P_f^i and P_i^{j+1} are proved by the following conditions [75]-

1. Points P_f^i and P_i^{j+1} are invariant points if, their Euclidean distance is lower than ε_{max} . Hence the resultant invariant point is the weighted average of P_f^i and P_i^{j+1} .
2. If condition 1 is false, P_f^i is the invariant point and P_i^{j+1} is not with the following conditions-

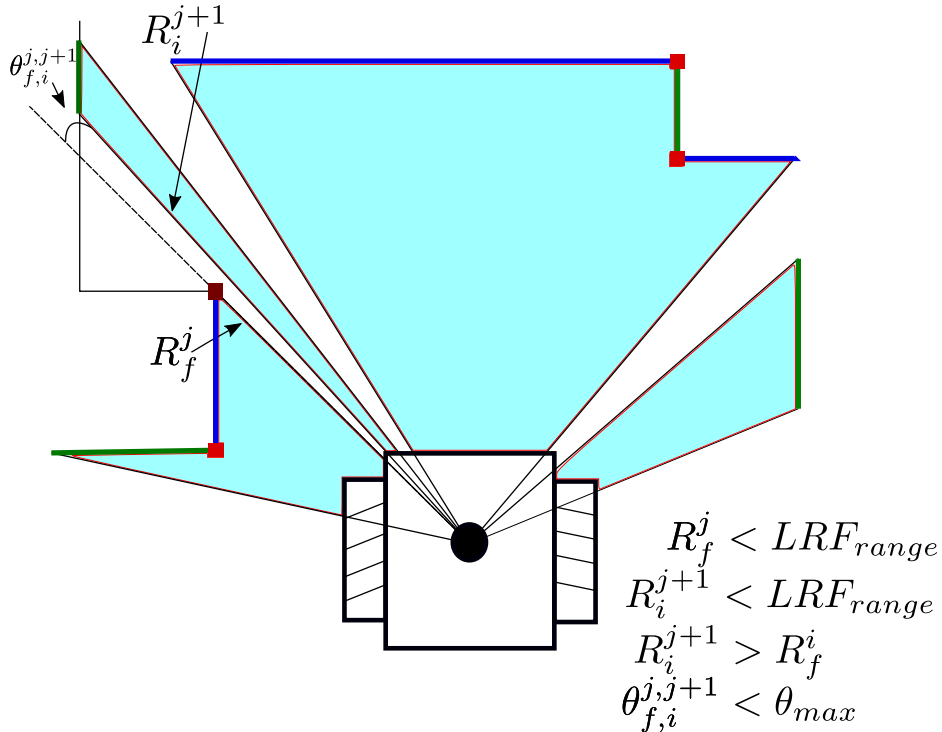


Fig. 3.7 Invariant point description. Invariant point (red and brown squares). The one with condition 1 are in red and condition 2 are in brown respectively.

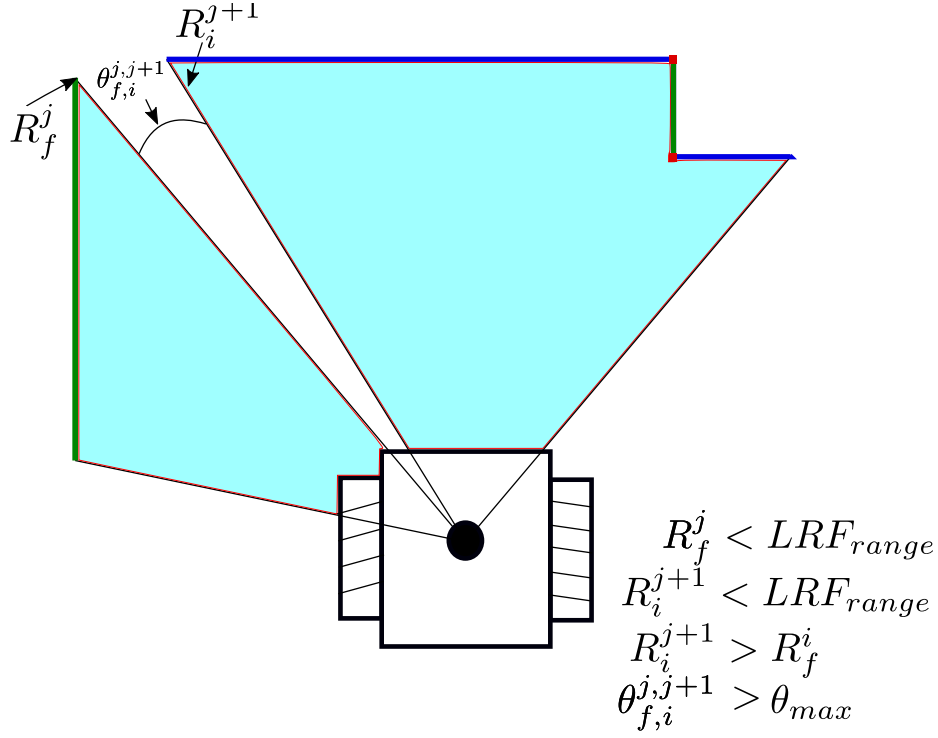


Fig. 3.8 Invariant point classification. The one with condition 1 are in red and condition 2 are in brown respectively. For points P_f^i and P_i^{j+1} the conditions is not satisfied.

- (a) R_f^i and R_i^{j+1} are lower than 8 metres (Range of LRF).
- (b) $R_i^{j+1} > R_f^i$.
- (c) Angle $\theta_{f,i}^{j,j+1}$ is below θ_{max} .

Figure 3.7 and Figure 3.8, the detected line segments are shown in blue and green. The invariant points are represented in square shape. The one with condition 1 are in red and condition 2 are in brown respectively. The resultant linear segments and invariant points are stored and their covariance and parameters are extracted in the data extraction mode.

The parameters for the split and merge criteria are described in Table 3.3 and Table 3.4. Same parameters as in the work by [75] are used here.

Table 3.3 Split and Merge parameters

Parameter	Value
ε_{merge}	0.070
ε_{split}	0.070
μ_{merge}	170

Table 3.4 Invariant point classification parameters

Parameter	Value
$\varepsilon_{max}(\text{metres})$	0.070
$\theta_{max}(\text{deg})$	3.5

3.4 Data Extraction

In the previous section, linear segments and invariant points from the LRF data were extracted. To use this information into the EKF, their parameters and covariance matrices are required. To find the line segment parameters and covariances, the standard Least Square technique is used. In the next chapter a new technique for extracting line segments by clustering the Hough space parameters will be explained which can extract lines even in noisy environments.

3.4.1 Line Segment Extraction

Consider a linear segment $Lin_{P_i,f} \in [P_i, \dots, P_j, \dots, P_f]$ from the LRF with $P_j = [x_j, y_j]$, in the mobile frame of reference. Then we can define the matrices

$$Y_{Lin} = [y_i \ \dots \ y_f]^T \quad (3.1)$$

$$X_{Lin} = \begin{bmatrix} x_j & \dots & x_f \\ 1 & \dots & 1 \end{bmatrix} \quad (3.2)$$

The cartesian presentation of the line is given by

$$y_j = mx_j + c \quad (3.3)$$

with slope m and intercept c with the y axis. It can be written as

$$g_{m,c} = \begin{bmatrix} m \\ c \end{bmatrix} \quad (3.4)$$

The line parameters can be rewritten as

$$Y_{Lin} = X_{Lin} g_{m,c} \quad (3.5)$$

The above linear equation can be solved using the Least Square Quadratic Error (LSQ) and the above parameters can be written as

$$X_{Lin}^T Y_{Lin} = X_{Lin}^T X_{Lin} g_{m,c} \quad (3.6)$$

$$g_{m,c} = \left[X_{Lin}^T X_{Lin} \right]^{-1} X_{Lin}^T Y_{Lin} \quad (3.7)$$

The LSQ for the best fitting parameters minimizes the following function

$$f_{Lin} = \sum_{j=1}^f (y_j - \hat{y}_j)^2 \quad (3.8)$$

where $\hat{y}_j = mx_j + c$. The estimated matrix $\hat{Y}_{Lin}$ can be defined as

$$\hat{Y}_{Lin} = [\hat{y}_i \quad \cdots \quad \hat{y}_f]^T \quad (3.9)$$

$$\hat{Y}_{Lin} = [mx_i + c \quad \cdots \quad mx_f + c]^T \quad (3.10)$$

The variance of vector Y_{Lin} is given by

$$\Sigma_{Y_{Lin}} = \frac{f_{Lin}}{f - 1} \quad (3.11)$$

in which f and i are the final and initial line segment indexes respectively. To calculate the matrix $g_{m,c}$, it can be described as the function Y_{Lin} using the Taylor expansion around the estimated vector $\hat{Y}_{Lin}$ by the following equation

$$g_{m,c}(Y_{Lin}) = g_{m,c}(\hat{Y}_{Lin}) + \nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin}) + \text{higher order terms} \quad (3.12)$$

Neglecting the higher order terms, $g_{m,c}$ as a function of Y_{Lin} is given approximately by

$$g_{m,c}(Y_{Lin}) \approx g_{m,c}(\hat{Y}_{Lin}) + \nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin}) \quad (3.13)$$

The variance of $g_{m,c}$ defined by $\Sigma_{m,c}$ can be calculated as

$$\begin{aligned}
 \Sigma_{m,c} &\approx E[(g_{m,c}(Y_{Lin}) - g_{m,c}(\hat{Y}_{Lin}))(g_{m,c}(Y_{Lin}) - g_{m,c}(\hat{Y}_{Lin}))^T] \\
 \Sigma_{m,c} &\approx E[(\nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin})(\nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin}))^T] \\
 \Sigma_{m,c} &\approx E[(\nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin})(Y_{Lin} - \hat{Y}_{Lin})^T \nabla g_{m,c}^T] \\
 \Sigma_{m,c} &\approx \nabla g_{m,c} E[(\nabla g_{m,c}(Y_{Lin} - \hat{Y}_{Lin})(Y_{Lin} - \hat{Y}_{Lin})^T] \nabla g_{m,c}^T \\
 \Sigma_{m,c} &= \nabla g_{m,c} \Sigma_{Y_{Lin}} \nabla g_{m,c}^T
 \end{aligned} \tag{3.14}$$

Here, $E[.]$ is the expected value of $[.]$. The gradient of $g_{m,c}$ in order of the the Y_{Lin} is given by $\nabla g_{m,c}$ and is given by

$$\nabla g_{m,c} = \frac{\partial g_{m,c}}{\partial Y_{Lin}} \tag{3.15}$$

From Equation (3.7),

$$\nabla g_{m,c} = [X_{Lin}^T X_{Lin}]^{-1} X_{Lin}^T \tag{3.16}$$

Therefore, the variance matrix $\Sigma_{m,c}$ can be written as

$$\Sigma_{m,c} = [X_{Lin}^T X_{Lin}]^{-1} \Sigma_{Y_{Lin}} \tag{3.17}$$

In order to avoid infinite slope m condition during the localization, the line segments are represented in the polar form instead of the cartesian form and are expressed by [75]

$$g_{\rho,\lambda} = \begin{bmatrix} \rho \\ \lambda \end{bmatrix} \tag{3.18}$$

$$\rho = x_j \cos \lambda + y_j \sin \lambda \tag{3.19}$$

where, ρ is the perpendicular distance relative to the origin of the mobile robot frame and λ is the angle between the perpendicular line segment to the line and the x_r axis, as shown in Figure 3.9 . The point (x_j, y_j) is also presented. The parameter ρ is given

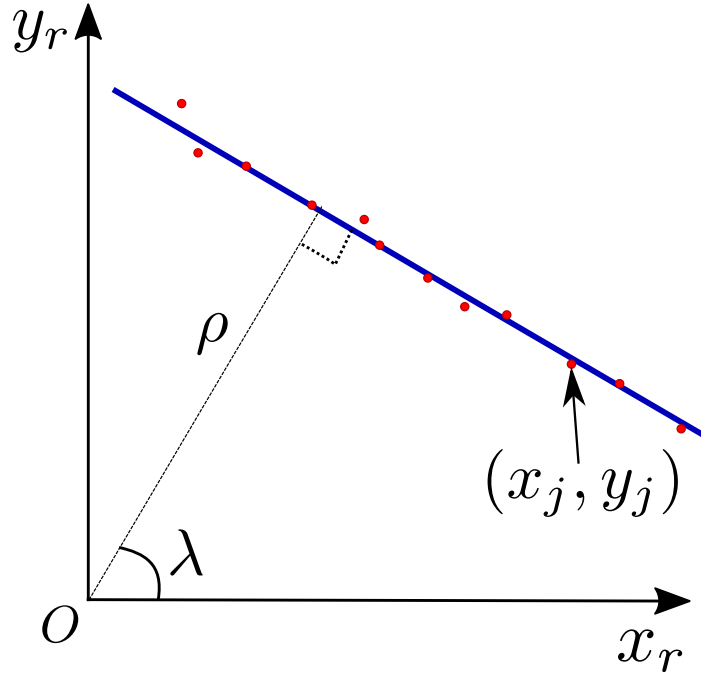


Fig. 3.9 Line presentation in polar form

by

$$\begin{aligned}
 \rho &= \min(\sqrt{x_j^2 + y_j^2}) \\
 \rho &= \min(\sqrt{x_j^2 + (m \cdot x_j + c)^2}) \\
 \rho &= \min(\sqrt{x_j^2(m+1) + 2m \cdot c \cdot x_j + c^2})
 \end{aligned} \tag{3.20}$$

Equation (3.20) is obtained when $\frac{df}{dx_j} = 0$, in which f is

$$f = x_j^2(m+1) + 2m \cdot c \cdot x_j + c^2 \tag{3.21}$$

The solution of $\frac{df}{dx_j} = 0$ appears when x_j and y_j are equal to

$$x_j = -\frac{m \cdot c}{m^2 + 1}, \quad y_j = \frac{c}{m^2 + 1} \tag{3.22}$$

Therefore, using Equation (3.21) and (3.22), the perpendicular distance ρ can be obtained using

$$\rho = \sqrt{\left(\frac{-m \cdot c}{m^2 + 1}\right)^2 + \left(\frac{c}{m^2 + 1}\right)^2} = \rho = \frac{c}{\sqrt{k^2 + 1}} \tag{3.23}$$

Angle λ is obtained by

$$\lambda = \tan^{-1} \left(\frac{\frac{c}{m^2+1}}{-\frac{m \cdot c}{m^2+1}} \right) = \tan^{-1} \left(-\frac{1}{m} \right) \quad (3.24)$$

Since the parameter $\rho > 0$, the line segment parameters in polar form can be represented as

$$g_{\rho,\lambda} = \begin{bmatrix} \frac{|c|}{\sqrt{m^2+1}} \\ \tan^{-1} \left(-\frac{\text{sign}(c)}{m} \right) \end{bmatrix} \quad (3.25)$$

in which $\text{sign}(c)$ represents the signal of c . The variance of $g_{\rho,\lambda}$, defined as $\Sigma_{\rho,\lambda}$ (from Equation (3.7)) can now be rewritten as a function of $g_{\rho,\lambda}$.

$$\Sigma_{\rho,\lambda} \approx \nabla g_{\rho,\lambda} \Sigma_{m,c} \nabla g_{\rho,\lambda}^T \quad (3.26)$$

From Equation (3.17), it can be simplified as

$$\Sigma_{\rho,\lambda} \approx \nabla g_{\rho,\lambda} [X_{Lin}^T X_{Lin}]^{-1} \nabla g_{\rho,\lambda}^T \Sigma_{Y_{Lin}} \quad (3.27)$$

where, $\nabla g_{\rho,\lambda}$ is the gradient of $g_{\rho,\lambda}$ given by

$$\nabla g_{\rho,\lambda} = \begin{bmatrix} \frac{\partial \rho}{\partial m} & \frac{\partial \rho}{\partial c} \\ \frac{\partial \lambda}{\partial m} & \frac{\partial \lambda}{\partial c} \end{bmatrix} \quad (3.28)$$

$$\nabla g_{\rho,\lambda} = \begin{bmatrix} -\frac{k \cdot c}{(m^2+1)^{\frac{3}{2}}} & \frac{1}{m^2+1} \\ \frac{1}{m^2+1} & 0 \end{bmatrix} \quad (3.29)$$

To sum up, the line segment parameters in the polar form with respect to robot frame are computed through Equation (3.25), while the covariance matrix can be obtained by Equation (3.27).

$$\Sigma_{\rho,\lambda} = \begin{bmatrix} \sigma_\rho^2 & \sigma_{\rho\lambda} \\ \sigma_{\lambda\rho} & \sigma_\lambda^2 \end{bmatrix} \quad (3.30)$$

$$\Sigma_{\rho,\lambda} \approx \begin{bmatrix} -\frac{m \cdot c}{(m^2+1)^{\frac{3}{2}}} & \frac{1}{m^2+1} \\ \frac{1}{m^2+1} & 0 \end{bmatrix} [X_{Lin}^T X_{Lin}]^{-1} \begin{bmatrix} -\frac{m \cdot c}{(m^2+1)^{\frac{3}{2}}} & \frac{1}{m^2+1} \\ \frac{1}{m^2+1} & 0 \end{bmatrix} \Sigma_{Y_{Lin}} \quad (3.31)$$

3.4.2 Corner Extraction

A corner formed by two intersecting lines P_j , is represented as an invariant point. The corner point is computed through the laser range R_j and the angle between the corner feature and robot frame θ_i as given in Figure and given by following equations:

$$P_j = (R_j + \varepsilon_{R_j}) \begin{bmatrix} \cos(\theta_j + \varepsilon_\theta) \\ \sin(\theta_j + \varepsilon_\theta) \end{bmatrix} \quad (3.32)$$

where point P_j is affected by noise in distance and angle given by ε_{R_j} and ε_{θ_j} . The error of the distance measurement ε_{R_j} is modeled by linear equation

$$\varepsilon_{R_j} = \varepsilon_m R_j + \varepsilon_c \quad (3.33)$$

where ε_m and ε_c are modeled as Gaussian noise with zero mean and standard deviation equal to σ_m and σ_c respectively. The error ε_θ is modeled as Gaussian noise with zero mean and standard deviation equal to σ_θ . The noise measured point P_j can then be written as

$$\begin{aligned} P_j &= (R_j + \varepsilon_m R_j + \varepsilon_c) \begin{bmatrix} \cos(\theta_j + \varepsilon_\theta) \\ \sin(\theta_j + \varepsilon_\theta) \end{bmatrix} \\ P_j &= (R_j + \varepsilon_m R_j + \varepsilon_c) \begin{bmatrix} \cos \theta_j \cos \varepsilon_\theta - \sin \theta_j \sin \varepsilon_\theta \\ \cos \theta_j \sin \varepsilon_\theta + \sin \theta_j \cos \varepsilon_\theta \end{bmatrix} \end{aligned} \quad (3.34)$$

Assuming $\cos \varepsilon_\theta \approx 1$ and $\sin \varepsilon_\theta \approx \varepsilon_\theta$, the observed noisy point becomes

$$P_j = (R_j + \varepsilon_m R_j + \varepsilon_c) \begin{bmatrix} \cos \theta_j - \varepsilon_{\theta_j} \sin \theta_j \\ \varepsilon_{\theta_j} \cos \theta_j + \sin \theta_j \end{bmatrix} \quad (3.35)$$

The estimated observed point $\hat{P}_j$ can be found by

$$\hat{P}_j = R_j \begin{bmatrix} \cos \theta_j \\ \sin \theta_j \end{bmatrix} \quad (3.36)$$

From Equation (3.35) and (3.36), the observed error $P_j - \hat{P}_j$ can be estimated by

$$P_j - \hat{P}_j = R_j \varepsilon_\theta \begin{bmatrix} -\sin \theta_j \\ \cos \theta_j \end{bmatrix} + (\varepsilon_m R_j + \varepsilon_c) \begin{bmatrix} \cos \theta_j \\ \sin \theta_j \end{bmatrix} \quad (3.37)$$

The variance of P_j can be computed as

$$\begin{aligned}\Sigma_{P_j} &= E[(P_j - \hat{P}_j)(P_j - \hat{P}_j)^T] \\ \Sigma_{P_j} &= E[R_j^2 \varepsilon_\theta^2 R_1] + E[(\varepsilon_m R_j + \varepsilon_c)^2 R_2] - E[2R_j \varepsilon_\theta (\varepsilon_m R_j + \varepsilon_c)] R_1 \\ \Sigma_{P_j} &= R_j^2 E[\varepsilon_\theta^2] R_1 + E[(\varepsilon_m R_j + \varepsilon_c)^2] R_2 - 2R_j E[\varepsilon_\theta (\varepsilon_m R_j + \varepsilon_c)] R_1\end{aligned}\quad (3.38)$$

where matrices R_1 , R_2 and R_3 are given by

$$R_1 = \begin{bmatrix} -\sin \theta_j \\ \cos \theta_j \end{bmatrix} \begin{bmatrix} -\sin \theta_i \\ \cos \theta_j \end{bmatrix}^T, R_2 = \begin{bmatrix} \cos \theta_j \\ \sin \theta_j \end{bmatrix} \begin{bmatrix} \cos \theta_j \\ \sin \theta_j \end{bmatrix}^T, R_3 = \begin{bmatrix} -\sin \theta_j \\ \cos \theta_j \end{bmatrix} \begin{bmatrix} \cos \theta_j \\ \sin \theta_i \end{bmatrix}^T \quad (3.39)$$

The expectation values are given by $E[\varepsilon_m] = 0$, $E[\varepsilon_c] = 0$, $E[\varepsilon_\theta]$ and $E[\varepsilon_\theta^2] = \sigma_\theta^2$, $E[\varepsilon_m^2] = \sigma_m^2$ and $E[\varepsilon_c^2] = \sigma_c^2$. ε_θ , ε_m and ε_c are Gaussian error with zero mean and covariance equal to σ_θ^2 , σ_m^2 and σ_c^2 , respectively. As the errors ε_θ , ε_m and ε_c are unrelated, expectation values of $E[\varepsilon_m \varepsilon_c]$, $E[\varepsilon_\theta \varepsilon_c]$ and $E[\varepsilon_\theta \varepsilon_m]$ are equal to zero. These can therefore be written as

$$E[(\varepsilon_m R_j + \varepsilon_c)^2] = R_j^2 E[\varepsilon_m^2] + 2R_j E[\varepsilon_m \varepsilon_c] + E[\varepsilon_c^2] \quad (3.40)$$

$$E[(\varepsilon_m R_j + \varepsilon_c)^2] = R_j^2 \sigma_m^2 + \sigma_c^2 \quad (3.41)$$

$$E[\varepsilon_\theta (\varepsilon_m R_j + \varepsilon_c)] = 0 \quad (3.42)$$

The observed feature P_j variance is equal to

$$\Sigma_{P_j} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} \quad (3.43)$$

$$\Sigma_{P_j} = R_j^2 \sigma_\theta^2 R_1 + (R_j^2 \sigma_m^2 + \sigma_c^2) R_2 \quad (3.44)$$

3.5 Conclusion

This chapter described different data extraction and detection methods for line segment extraction and invariant points from the LRF data. A standard split and merge algorithm is used which clusters the LRF scans into segments with scan points and merges the clusters into one when the threshold is met. The line extraction module uses least square method for extracting lines from the scans. The line segment points

and the invariant points are utilized in the EKF algorithm in the next chapter for map building and localization.

Chapter 4

Clustering Approaches to Mapping

4.1 Introduction

In the last chapters, probabilistic frame work for the robot SLAM problem and data extraction techniques from the LRF data were discussed in detail. This chapter concretely discusses the applications of clustering techniques for robot map building in indoor scenarios. In this chapter, noise removing techniques, data clustering techniques and application of these techniques to line extraction from LRF data will be discussed.

Clustering or cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal of data analysis is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity within a group and the greater the difference between groups, the better or more distinct the clustering.

Figure 4.1 shows an example of clustering result applied on a set of data points. Depending on the user input and application different number of clusters can be taken from the sample data set. The clustered data set can then be used for different applications[30, 42, 43].

4.2 LRF data clustering in spatial domain

This section describes LRF data clustering in the spatial domain for robot mapping. A spatial domain represents LRF data in space (i.e., x, y coordinates for the 2D case) with the direct manipulation of the LRF data during its processing. Clustering (ex. k -means clustering) is sensitive to noise [56], which may result in inaccurate cluster formation. Therefore, it is necessary to first remove noise from the LRF data and then

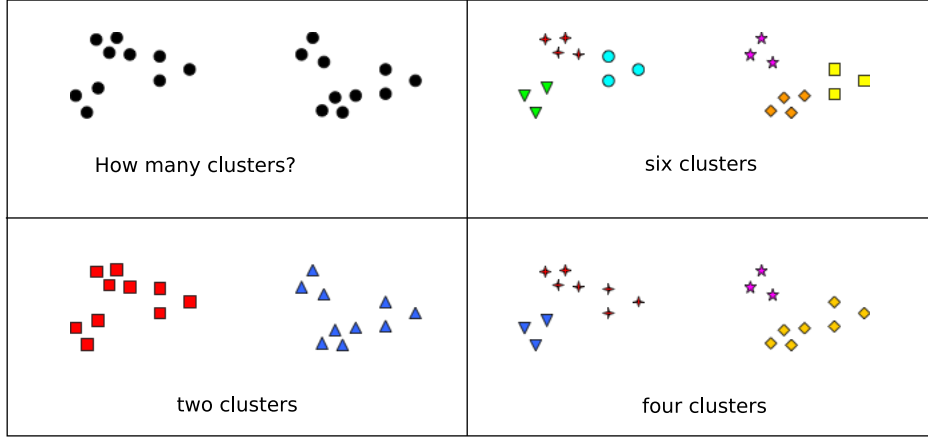


Fig. 4.1 Different ways of clustering the same set of data points [97]

apply clustering. The robot odometry model is described in Appendix 1 and from the odometry model we obtain the Δx , Δy and $\Delta \theta$.

A robot with a LRF mounted on it moves in small *steps*, which are the robot's displacement in *step-time* $\Delta s_{t_n} = t_n - t_{n-1}$ within the two time intervals t_{n-1} and t_n . From Equation (A.11) (Appendix 1), for the robot's movement perpendicular to the x -axis, $\theta = 90$ degrees and displacement $\Delta x = 0$, while displacement Δy is given by:

$$\Delta y = \frac{(D_r(t_n) - D_r(t_{n-1})) + (D_l(t_n) - D_l(t_{n-1}))}{2}, \quad (4.1)$$

Once the data is read from the LRF, it is grouped into segments $S^i, i > 0$, as shown in Figure 4.2, using the data segmentation techniques described in Section 4.2.2. In each step-interval Δs_t , the raw LRF data obtained is segmented. Later, techniques for noise removal or clustering are applied to each segment.

4.2.1 Modes of LRF clustering

We define the following three modes for map generation after the noise reduction and clustering of the raw LRF data to generate maps:

1. *Online Mode*: In Online Mode, the robot continuously applies noise reduction and clustering on the data as soon as they are available with each scan after the step-interval Δs_t . Figure 4.3a shows the working of the robot in *Online Mode*. The raw LRF data measured at each step-interval $\Delta s_{t_n} = t_n - t_{n-1}$ is shown in Figure 4.3a-(i). In each step-interval Δs_t , two operations are performed. First, noise is removed by the methods discussed in Section 4.3.1 and/or Section 4.3.3. The noise removed at each Δs_t is shown in Figure 4.3a-(ii). Later, in

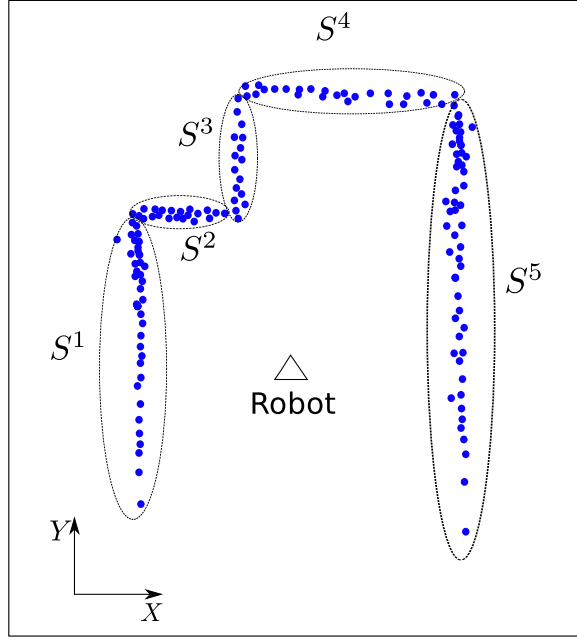
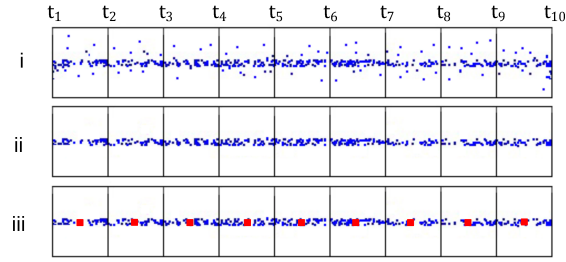


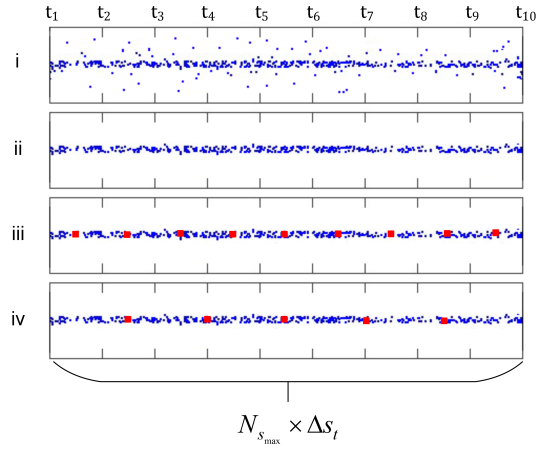
Fig. 4.2 LRF data segmentation.

the same step-interval, clustering is performed, as described in Section 4.5, to obtain single or multiple clusters. This is shown in Figure 4.3a-(iii) with the red dots representing the centroids of individual clusters. Generally, any given single cluster (and hence a single centroid) obtained in each step-interval as raw LRF data in Δs_t is small. However, multiple clusters (and hence multiple centroids) can also be obtained if the raw LRF in Δs_t is large enough. The data which has been processed once in a step-interval Δs_{t_n} is marked as ‘*processed*’ and is not processed again in the next step-interval $\Delta s_{t_{n+1}}$. Thus, in $\Delta s_{t_{n+1}}$, noise removal and clustering is applied on newly obtained data and the process is repeated.

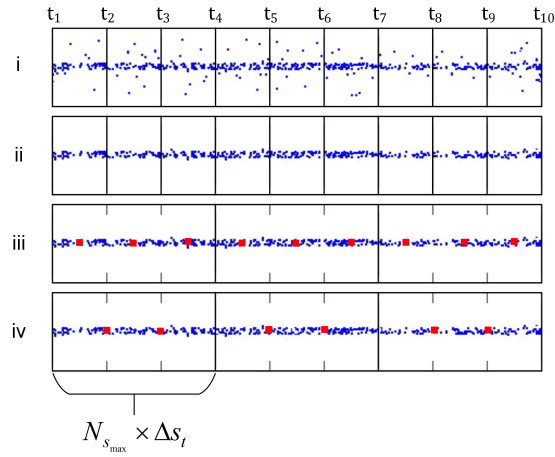
2. *Offline Mode*: In this mode, raw LRF data is accumulated until a predefined $N_{s_{max}}$ time-steps (Δs_t), and noise reduction and clustering is applied on the accumulated (*offline*) LRF data after $N_{s_{max}} \times \Delta s_t$ steps for a given θ . This is shown in Figure 4.3b. *Offline Mode* is useful when *step-time* Δs_t produces very little data, which is inefficient for clustering. The raw LRF data is shown in Figure 4.3b-(i) obtained at each step-interval $\Delta s_{t_n} = t_n - t_{n-1}$. Here, $N_{s_{max}}$ is set to ‘9’. Unlike *Online Mode*, two operations of noise removal and clustering are applied to the accumulated LRF data after $9 \times \Delta s_t$ step-intervals. Figure 4.3b-(i) shows the accumulated LRF data with $N_{s_{max}} = 9$. Figure 4.3b-(ii) shows the results of the noise removal. Figure 4.3b-(iii) shows nine centroids obtained after



(a) Online mode



(b) Offline mode.



(c) Mixed mode

Fig. 4.3 Modes of clustering. (a) Online Mode, (b) Offline Mode, and (c) Mixed Mode. The red dots represent the centroid obtained after clustering.

the clustering of the offline data. Notice that the number of clusters of offline data may vary. For example, Figure 4.3b-(iv) shows five centroids obtained after the clustering of noise-free data shown in Figure 4.3b-(ii). However, if the robot changes its direction (i.e., for a change in θ), noise removal and clustering are applied after $n \times \Delta s_t$, even if $n < N_{s_{max}}$, to eliminate data processing for the special case when θ changes during accumulation.

3. *Mixed Mode*: In Mixed Mode, some of the LRF data processing takes place in *Online Mode*, whereas others take place in *Offline Mode*. For example, noise reduction and clustering can take place in *Online Mode* and *Offline Mode*, respectively. This is shown in Figure 4.3c. The LRF data is scanned at each step-interval as shown in Figure 4.3c-(i). Noise is removed after each step-interval, as shown in Figure 4.3c-(ii). However, clustering is performed on accumulated (*offline*) data after $3 \times \Delta s_t$ step-intervals ($N_{s_{max}} = 3$). Three clusters are made offline, as shown in Figure 4.3c-(iii), whereas two clusters are shown in Figure 4.3c-(iv).

4.2.2 LRF Data Segmentation

Prior to extracting lines, the LRF data is pre-processed into distinct segments. The LRF scans the environment and acquires n scan points, represented as:

$$P_i = [x_i, y_i] = [r_i \cos \varphi_i, r_i \sin \varphi_i], \quad i = 1, 2, \dots, n \quad (4.2)$$

where r_i is the distance between the i^{th} scanned point with respect to the origin of the LRF, and φ_i is the angle of the i^{th} scan point with respect to the X_L axis of the LRF frame given by $O - X_L Y_L$ in Figure 4.4.

Initially, the standard split and merge technique [71] is used for segmenting the data without line extraction for its robustness and speed which is discussed in Chapter 3. In the split process, the adaptive breakpoint detector (ABD) method to detect any discontinuity in the data [13, 76] is used. This can make inferences about the possible presence of discontinuities in a sequence of valid range data. In the real world, such discontinuity accounts for open doors, windows, open passages and other areas which it is important to recognize. A point distance-based segmentation (PDBS) method detects these breakpoints by finding the Euclidean distance between the two

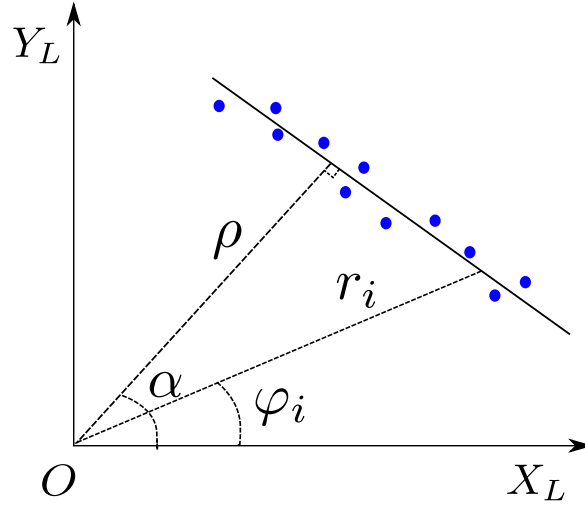


Fig. 4.4 Geometry of a scanned point and a corresponding line.

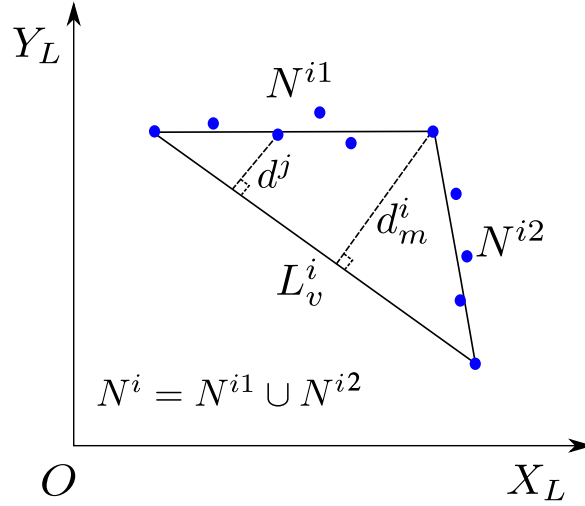


Fig. 4.5 The principle behind the IEPF algorithm.

consecutive points P_i and P_{i+1} :

$$\|P_{i+1} - P_i\| < D_{th} \quad (4.3)$$

ABD uses the distance threshold D_{th} via the following:

$$D_{th} = r_i \frac{\sin(\Delta\phi_a)}{\sin(\lambda - \Delta\phi_a)} + 3\sigma_r \quad (4.4)$$

where $\Delta\phi_a$ is the angular resolution of the LRF and σ_r is the residual variance to encompass the stochastic behaviour of the sequence of the scanned points p_i and the

related noise associated with r_i . The deviation is provided by the LRF manufacturer and λ is an auxiliary parameter. The value of λ has been found to perform well when setting the value as $\lambda = 10^\circ$.

After getting the breakpoints, the clusters are split using the iterative end point fitting (IEPF) algorithm [24] and segments S^i are obtained. IEPF is a recursive algorithm for line extraction and for finding clusters. The principle of the IEPF algorithm is shown in Figure 4.5. For a cluster of points given by N_i , it uses a virtual line L_v^i between two end points of the cluster. It then calculates the distance of every point d_j to the line L_v^i and finds the maximum distance d_m^i . If $d_m^i > D_T$, the IEPF algorithm splits the cluster of points N^i into two subsets N^{i1} and N^{i2} , where D_T is a predefined splitting threshold. The process is iterated to each subset until no new subset can be found. The value of D_T is generally chosen to be small, but for long-range sensors the data gets noisy as the range increases and it affects the splitting. A modified iterative end point fitting (MIEPF) algorithm is utilized to set the value of D_T [63, 64]. The value is set as:

$$D_T = 3\sigma_{r_0} + \epsilon_a(\rho_h - r_0) \quad (4.5)$$

where ϵ_a is an artificial setting ratio that accounts for the performance of the LRF and the user's experience. σ_{r_0} is the variance of the measurement at range r_0 . The value of $D_{T_{max}}$ is set to restrict the loosing threshold based on the real performance of the LRF. Figure 4.2 shows the segmentation of the raw LRF data using the process described above to obtain segments $(S^1, S^2, \dots, S^5)$.

4.3 Noise Removing Techniques

Although LRF data is mostly accurate in structured environment with orthogonal features (straight walls, doors, corridors), in reality, most of the indoor structures are not empty and are cluttered with objects (furniture, boxes, tables, chairs, dust bins and other objects). These objects and the nature of the level of clutters causes the LRF data to produce what is called high sensor noise. These noise is important to be removed prior to mapping and localization for accurate feature extraction. In the subsequent sections, application of clustering for removing the sensor noise are discussed.

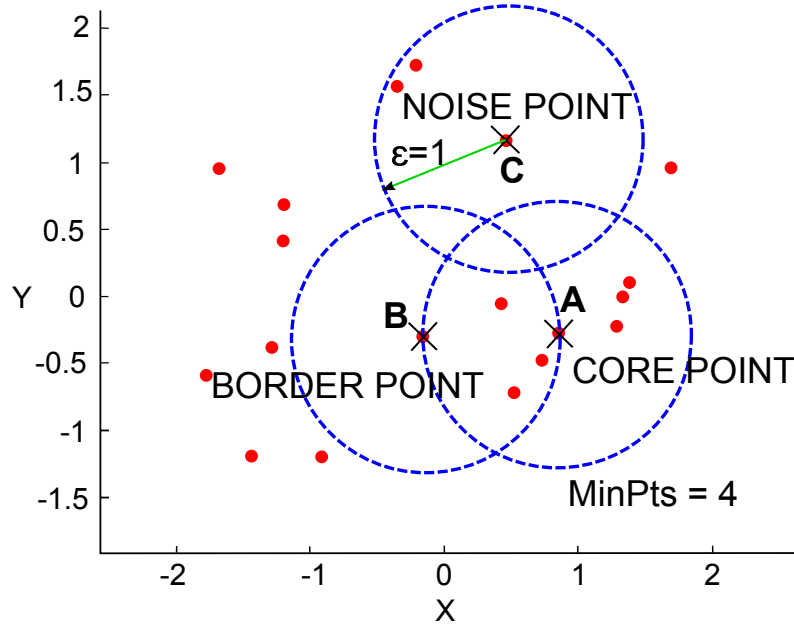


Fig. 4.6 Core, border points and noise points in center-based density.

4.3.1 DBSCAN

The density in the center-based approach is estimated for a particular point in the data set by counting the number of points within a specified radius Eps , of that point. It classifies a point as [28, 48, 97]:

1. *in the interior of the dense (core) region.*
2. *on the edge of a dense region (border point).*
3. *in a sparsely occupied region (noise).*

Core Points: A point is a core point if the number of points within a given neighbourhood around the point as determined by the distance function and user-specified distance parameter, Eps , exceeds a certain threshold, $MinPts$, which is also a user-specified parameter. Figure 4.6, point A is a core point for $MinPts \leq 7$

Border Points: A border point is not a core point, but falls within the neighbourhood of a core point. A border point can fall within the neighbourhood of many core points. In Figure 4.6, point B is a border point.

Noise Points: A noise point is any point that is neither a core point nor a border point. In Figure 4.6, point C is a noise point. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is based on density reachability [28]. A point

q is directly density-reachable from a point p if it is not further away than a given distance ε (i.e., is part of its ε -neighborhood), and if p is surrounded by sufficiently many points such that p and q can be part of a cluster. The point q may be on the edge of the cluster leading to the term “density-connected”: two points p and q are density-connected if there is a point o such that o and p as well as o and q are density-reachable. Hence, all points within a cluster are mutually density-connected, and if a point is density-connected to any other point of the cluster, it is part of the cluster as well. The process to establish clustering starts at an arbitrary point (marked ‘visited’) and if there are sufficient to many points in the neighborhood ($< \varepsilon$), this initiates the clustering process, otherwise the point is marked as ‘noise’ and the procedure is repeated from/at another point until all the points have been visited. To understand more clearly the algorithm is formalized in notion of *clusters* and *noise* in a database D of points of some k -dimensional space S . The shape of the neighbourhood is determined by the choice of a distance function for two points p and q , denoted by $dist(p, q)$. To define the algorithm all examples are visualized in 2D Euclidean distance space [28].

Definition 1 (*Eps neighbourhood of a point*) A Eps neighbourhood of a point p , denoted by $N_{Eps}(p)$, is defined by,

$$N_{Eps}(p) = \{q \in D \mid dist(p, q) \leq Eps\}$$

There are two kinds of points in a cluster, points inside of the cluster (*core points*) and points on the border of the cluster (*border points*). In general the *Eps*-neighbourhood of a border point contains significantly less points than an *Eps*-neighborhood of a core point. In order to include all points belonging to the same cluster, the minimum number of points is set to a relatively low value. This value, however, will not be characteristics for the respective cluster particularly in the presence of noise. Therefore, we require that for every point p in a cluster C there is a point q in C so that p is inside of the *Eps*-neighbourhood of q and $N_{Eps}(q)$ contains at least *MinPts* points.

Definition 2 (*directly density-reachable*) A point p is directly density-reachable from a point q with respect to *Eps*, *MinPts* if

1. $p \in N_{Eps}(q)$ and
2. $|N_{Eps}(q)| \leq MinPts$

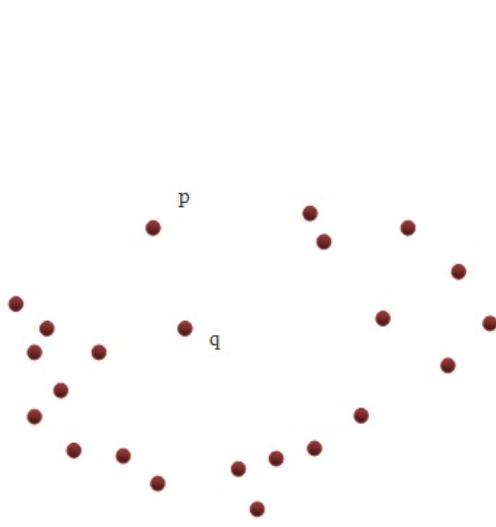


Fig. 4.7 p is border point, q is core point

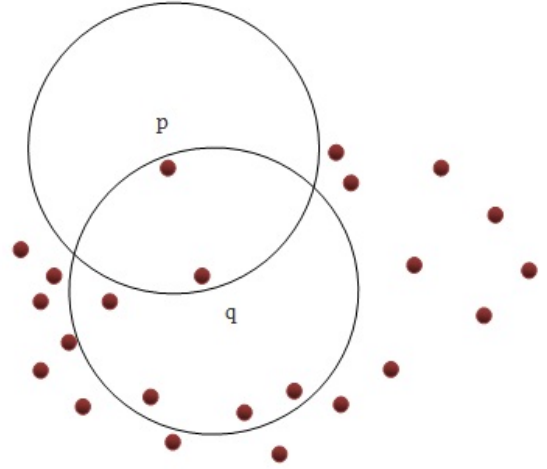


Fig. 4.8 Asymmetrical case: p directly density-reachable from q; q not directly density-reachable from p.

Directly density-reachable is symmetric for pairs of core points. However it is not symmetric if one core point and one border point are involved. Figure 4.7 and Figure 4.8 show the core and border point with asymmetrical case. Figure 4.8 show that point 'p' is directly density-reachable to point 'q' whereas; point 'q' is not directly density-reachable to point 'p'.

Definition 3 (*density-reachable*) A point p is density-reachable from a point q with respect to Eps and $MinPts$ if there is a chain of points $p_1, \dots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i .

Density-reachability is a canonical extension of direct density-reachability and, the relation is transitive but, not symmetric. Two border points of the same cluster C are possibly not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in C from which both border points of C are density-reachable. To cover the relation of border points, a notion of density-connectivity is defined next.

Definition 4 (*density-connected*) A point p is density-connected to a point q with respect to Eps and $MinPts$ if there is a point 'o' such that both, 'p' and 'q' are density-reachable from 'o' with respect to Eps and $MinPts$.

Density-connectivity is a symmetric relation. For density-reachable points, the relation of density-connectivity is also reflexive. Figure 4.9 shows how point 'p' and 'q' are

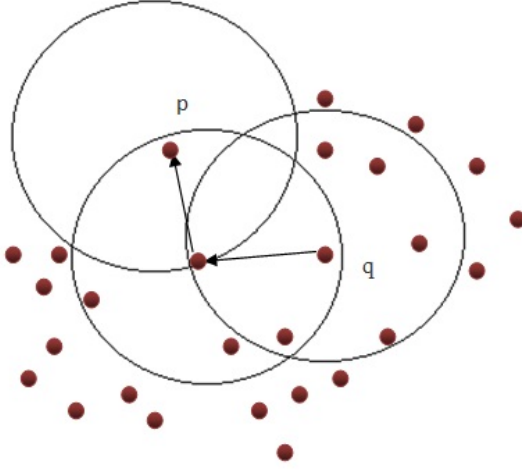


Fig. 4.9 Density-reachability. p is density-reachable from q ; q is not density-reachable from p .

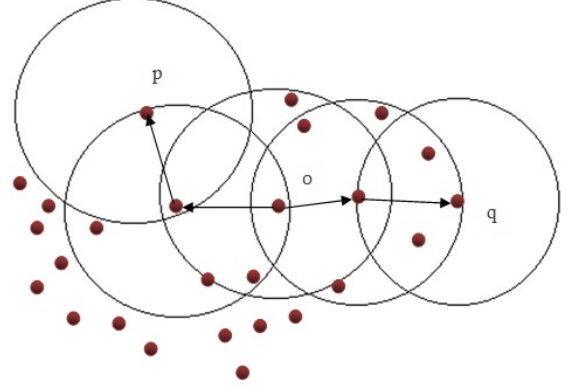


Fig. 4.10 Density connected points. p and q are density connected to each other by o

density-reachable with respect to each other. Figure 4.10 shows points ' p ' and ' q ' are density-connected to each other via point ' o '.

Definition 5 (*cluster*) Let D be a database of points. A cluster C wrt. Eps and $MinPts$ is a non-empty subset of D satisfying the following conditions:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt. Eps and $MinPts$, then $q \in C$. (*Maximality*)
2. $\forall p, q \in C$: p is density-connected to q wrt. Eps and $MinPts$. (*Connectivity*)

Definition 6 (*noise*) Let $C_1, \dots, C_k$ be the clusters of the database D wrt. Eps and $MinPts_i$, $i = 1, \dots, k$. Then we can define noise as the set of points in the database D not belonging to any cluster C_i , i.e.

$$noise = \{p \in D \mid \forall i : p \notin C_i\}$$

The DBSCAN algorithm is formally presented in Algorithm 2 [97]. Any two core points that are close enough within a distance Eps of one another are put in the same cluster. Any border point that is close enough to a core point is put in the same cluster as the core point. Noise points are discarded. Figure 4.12 shows an example of the laser scan data of the environment. The blue dots represent obstacle points that are detected by the LRF. While the robot was moving the scan shows areas that are actually not

present in the environment, which are circled by red shown in Figure 4.12a. Since the same entities or features in the environment were mapped again and again, the scan shows overlapped result of the same features. Figure 4.12b shows the result obtained after applying the DBSCAN algorithm on the raw data set. It effectively removes the noise areas. One drawback can be removal of actual data from the scans, but since the values of ϵ and $MinPts$ can be set freely, the user can come up with a good set of values for the algorithm to perform well.

Algorithm 1 DBSCAN Algorithm

- 1: Label all points as *core*, *border*, or, *noise* points.
 - 2: Eliminate noise points.
 - 3: Put an edge between all core points that are within Eps of each other.
 - 4: Make each group of connected core points into a separate cluster.
 - 5: Assign each *border* point to one of the clusters of its associated *core* points.
-

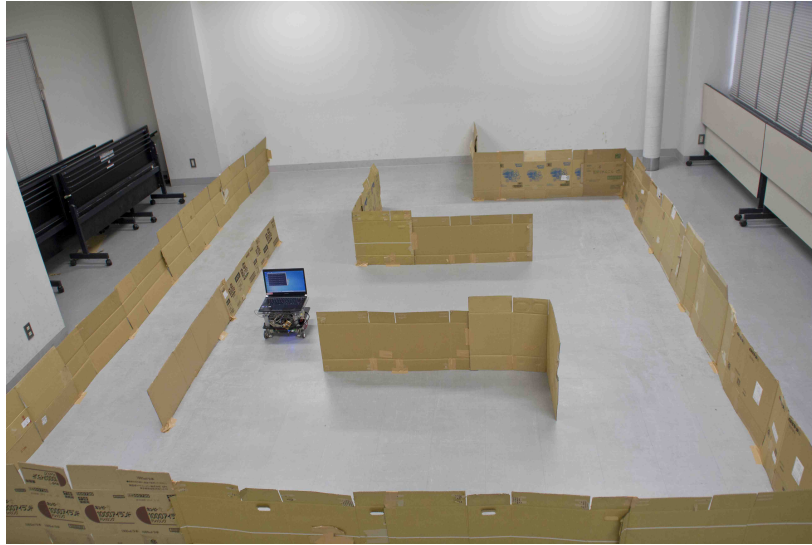
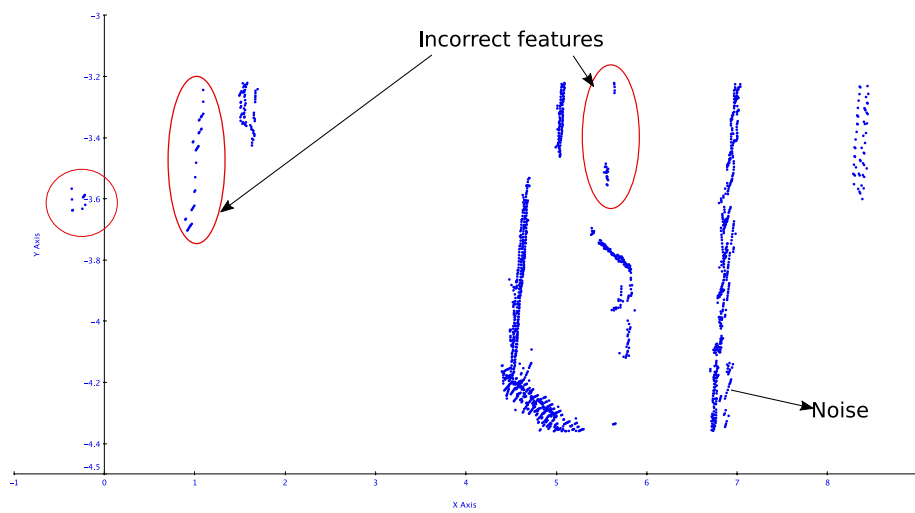


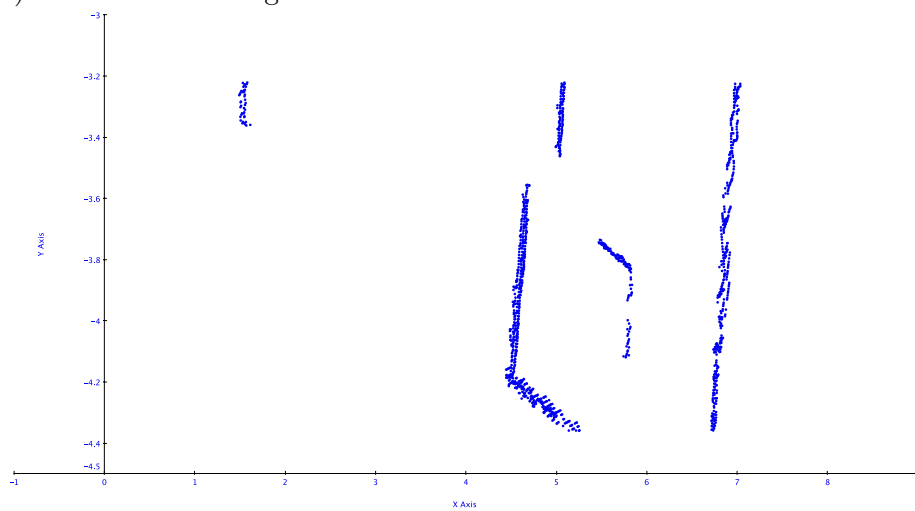
Fig. 4.11 Simple simulation environment.

4.3.2 Results- DBSCAN

The robot with the LRF was moved in an artificial test environment as shown in Figure 4.11. The values of ϵ and N_{MinPts} are set by training the robot and looking into the noise characteristics. The robot is trained in an actual environment based on user feedback. For a fixed value of N_{MinPts} , a maximum value is assigned to ϵ , i.e., $\epsilon = \epsilon_{max}$. The value of ϵ is consequently decreased - step by step - until a minimum ϵ_{min} is achieved, which gives an optimum noise reduction. Figure 4.11 shows the test

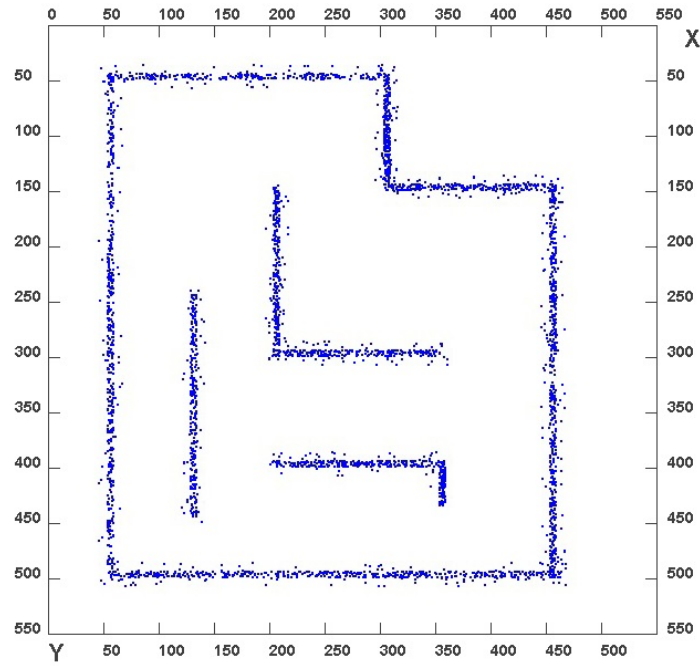


(a) Laser scan showing the raw LRF data with noise areas circled in red.

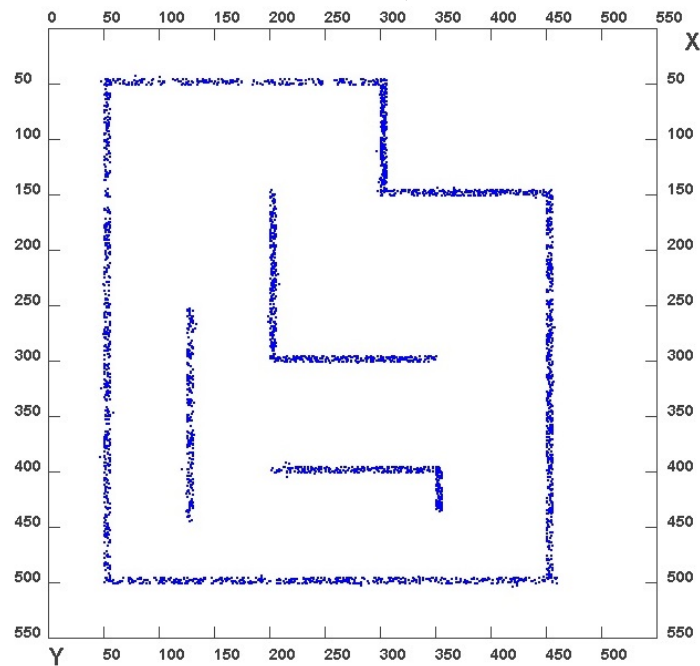


(b) DBSCAN result.

Fig. 4.12 DBSCAN result on sample LRF data.



(a) Raw laser scan with noise. (Random noise was added to test the DBSCAN algorithm)



(b) Noise removed using DBSCAN.

Fig. 4.13 DBSCAN result on the test environment.

environment where the robot was moved to collect the data points. Figure 4.13 shows the result of DBSCAN on the test environment where, Figure 4.13a shows the raw LRF data of the environment without removing noise. Figure 4.13b shows the result of DBSCAN applied on the raw data.

4.3.3 Noise removal using Mathematical Morphology

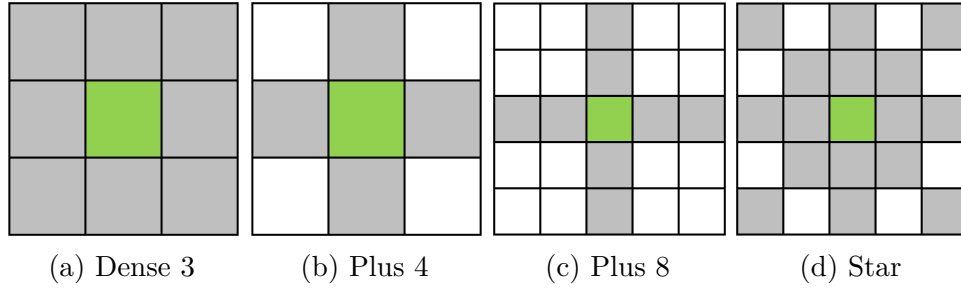


Fig. 4.14 Structuring element kernel type. (a) DENSE $\times 3$, (b) PLUS $\times 4$, (c) PLUS $\times 8$, (d) STAR $\times 8$. The central data point indicated in green is the point to be processed.

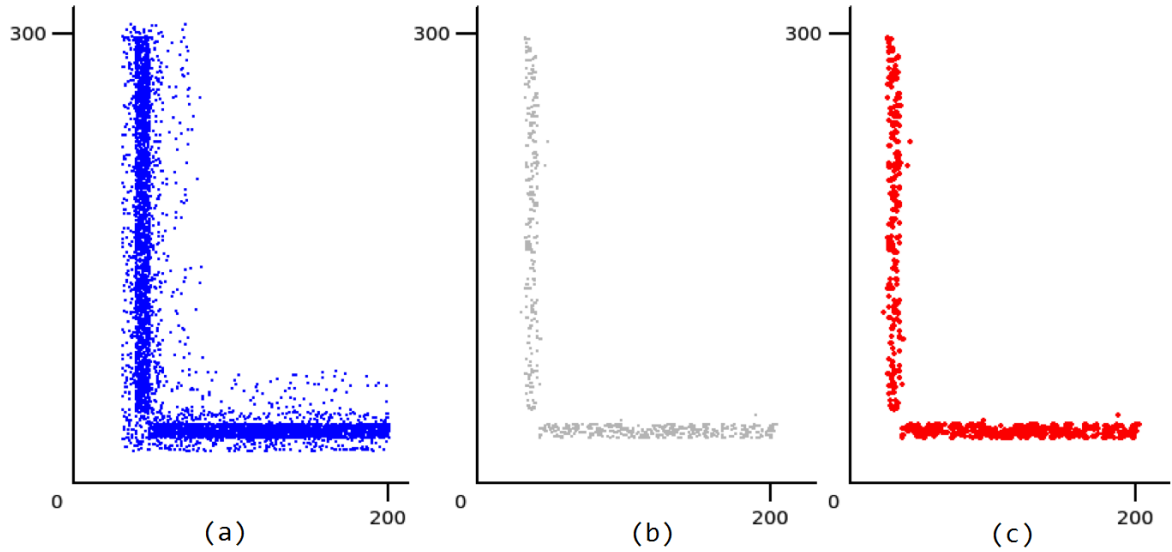


Fig. 4.15 Result of erode-dilate on LRF data.(a) Raw laser data of a wall with noise, (b) the erode operation, and (c) the dilate operation on (b).

Morphology is a branch of mathematics which deals with a wide range of operators and which is particularly useful for the analysis of binary images. This section shows the applicability of these techniques on LRF data. The basic idea is to treat each point $P_{x,y}$ obtained from the LRF as a pixel. Initially, the entire grid of size $height \times width$ of the setup environment is supposed to be *null* (i.e., zero). The points in the grid

where the LRF data have a value greater than zero are treated as one. Next, we apply an erosion algorithm over the untreated scanned data. Details of the erosion technique can be found in various sources [85, 89]. Erosion basically takes the logical *AND* operator of neighbouring data (which can be 0 or 1) to output the resultant data. In order to effectively apply an erosion technique over raw sensor data, this section describes different kinds of *kernels*, which can be selected by the user according to the characteristics of noise. Each kernel is characterized by three parameters given by a logical vector:

$$V = [\pm\psi_H \quad \pm\psi_V \quad \pm\psi_C]^T \quad (4.6)$$

Here, H, V and C represent the horizontal, vertical and crosswise (or diagonal) directions, respectively. Figure 4.14 shows the kernels DENSE $\times 3$, PLUS $\times 4$, PLUS $\times 8$ and STAR $\times 8$.

The raw LRF data (denoted by A) is viewed as a subset of the integer grid $E = \mathbb{Z}^{dim}$ where $dim = 2$. The raw data is probed with a predefined shape (structuring element, denoted by B) shown in Figure 4.14b, which is a binary datum itself whose shape is defined by vector ψ_i where $i \in H, V, C$. For example, the structuring element $B=PLUS\times 8$ (Figure 4.14c) is defined by:

$$\psi_i = \pm 2, i \in (H, V), \psi_C = 0 \quad (4.7)$$

$$E = \mathbb{Z}^2; B = \{(-2, 0), (-1, 0), (0, 0), (1, 0), (2, 0), \\ (0, -1), (0, -2), (0, 1), (0, 2)\} \quad (4.8)$$

Erosion is defined by [93]:

$$A \ominus B = \{z \in E | B_z \subseteq A\} \quad \text{or} \quad A \ominus B = \bigcap_{b \in B} A_{-b} \quad (4.9)$$

where B_z is the translation of vector B by vector z :

$$B_z = \{b + z | b \in B\}, \forall z \in E \quad (4.10)$$

Erosion may also remove the actual data along with the noise. Although this may seem to be a drawback, in the case of LRF data it actually helps in data reduction for the faster computation of clustering [81]. Some of the actual data lost by erosion can

Table 4.1 Definition of various kernels with parameters.

Fig.No.	Kernel Type	ψ_H	ψ_V	ψ_C
Figure 4.14a	DENSE $\times 3$	± 1	± 1	± 1
Figure 4.14b	PLUS $\times 4$	± 1	± 1	0
Figure 4.14c	PLUS $\times 8$	± 2	± 2	0
Figure 4.14d	STAR $\times 8$	± 2	± 2	± 2

be obtained by performing a dilate operation, which is defined by:

$$A \oplus B = \left\{ z \in E \mid (B^S)_z \cap A \neq \emptyset \right\} \quad \text{or} \quad A \oplus B = \bigcup_{b \in B} A_b \quad (4.11)$$

where B^S denotes the symmetric of B , i.e.:

$$B^S = \{x \in E \mid -x \in B\} \quad (4.12)$$

The kernels shown in Figure 4.14 are defined by the three parameters (ψ_H, ψ_V, ψ_C) summarized in Table 4.1. It is obvious that the kernel DENSE $\times 3$ will remove more noise than the kernel PLUS $\times 4$. Similarly, the kernel STAR $\times 8$ will remove more noise than the kernel PLUS $\times 8$. Since some actual LRF data also gets removed during the noise removal process, it is beneficial to actually try out and train the robot using different kernels in the actual environment. This can easily be done using the proposed framework, which is discussed in detail in Section 4.7. Figure 4.15(b) shows the results of noise reduction by Erosion using the PLUS $\times 4$ kernel on noisy LRF data shown in Figure 4.15(a). Notice that some of the actual LRF data also gets removed, which can be recovered by applying dilation (the result of which is shown in Figure 4.15(c)).

4.4 Embarrassingly Parallel Noise Removal Technique from LRF data

This section proposes an algorithm which is very fast and embarrassingly parallel [80]. In parallel computing, an embarrassingly parallel workload, or embarrassingly parallel problem, is one for which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case where there exists no dependency (or communication) between those parallel tasks [35]. While algorithms like DBSCAN scan each pixel or data element's neighbor to check the density, the proposed algorithm scans the entire input data using a window which can perform multiple hops to accelerate

processing. By performing multiple hops, acceleration is obtained by actually not processing some of the parts of the input data which are less significant in features. A ‘reverse-hop’ mechanism ensures that significant data is not omitted during multiple hops.

4.4.1 Hopping Noise Removal Algorithm

The algorithm is based on density of Lidar sensor data. Initially, the entire Lidar sensor data of the environment is treated as a binary image of size $(W \times H)$ where W and H represents the width and height of the environment, respectively. Each data point $D(x, y)$ where Lidar sensor has a value is treated as a *HIGH* pixel $P(x, y)$ of the image. Points where there is no Lidar sensor data is treated as a *LOW* pixel of the image. Here, *HIGH* and *LOW* represents pixel values 1 and 0, respectively (as the image is binary).

$$P(x_i, y_j) = \begin{cases} 1, & \text{if } D(x_i, y_j) \text{ exists} \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in (0 \cdots W), \forall j \in (0 \cdots H) \quad (4.13)$$

A hopping window $\Psi_{r,c}^{w,h}$ is a rectangular window within the image space of height h and width w , where, $(0 < h < H)$ and $(0 < w < W)$, with (r, c) representing the topmost coordinates of hopping window. $\Psi_{r,c}^{w,h}$ scans the entire image from $P(0, 0)$ to $P(H, W)$ and checks for the pixel density represented by δ as shown in Figure 4.16. Pixel density δ is the total number of *HIGH* pixels per square unit of total number of pixels in $\Psi_{r,c}^{w,h}$ and is given by the following equation,

$$\delta = \frac{\sum_{i=r}^{r+h} \sum_{j=c}^{c+w} P(x_i, y_j)}{w \times h} \quad (4.14)$$

where, $P(x_i, y_j)$ is given by Equation (4.13). If the pixel density in a hopping window $\Psi_{r,c}^{w,h}$ is less than a threshold density τ_δ , pixels in that window are considered to be noise and removed. Otherwise, the pixels in that window are treated as ‘data’.

$$P(x_i, y_j) = \begin{cases} \text{NOISE}, & \text{if } \delta < \tau_\delta \\ \text{DATA}, & \text{if } \delta \geq \tau_\delta \end{cases} \quad (4.15)$$

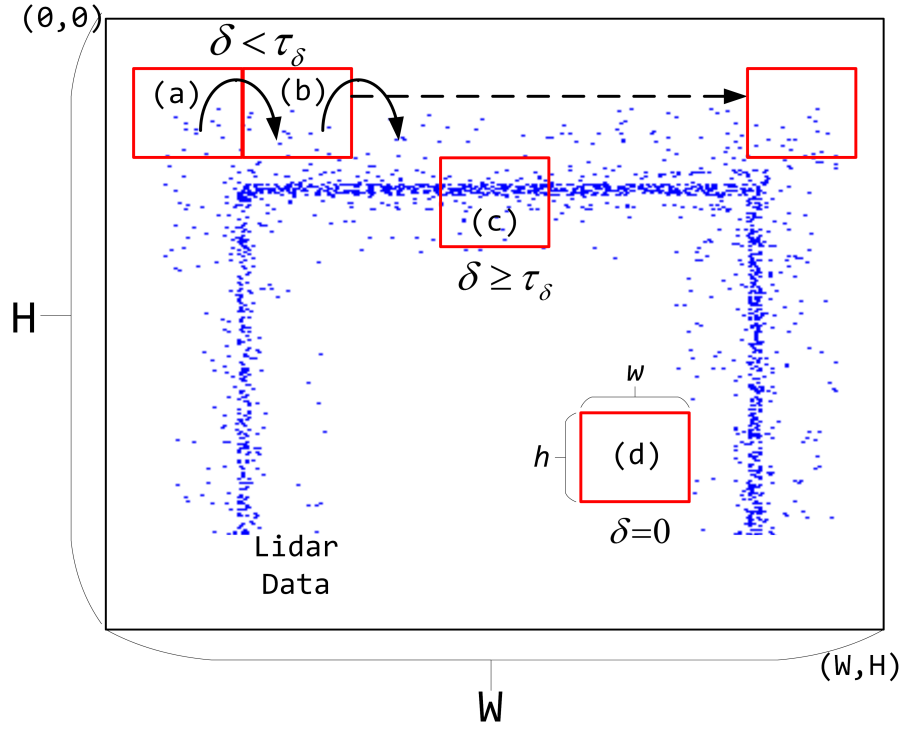


Fig. 4.16 Hopping window concept. (a) and (b) $\delta < \tau_\delta$ treated as noise, (c) $\delta \geq \tau_\delta$ treated as data, (d) $\delta = 0$, trigger for multiple hopping condition

In Figure 4.16, window Ψ shown by (a) and (b) satisfies $\delta < \tau_\delta$ and all the pixels inside the window are treated as noise and removed. For Ψ in (c), $\delta \geq \tau_\delta$ is satisfied and data is not removed. For Ψ shown in (d), $\delta = 0$ satisfies and it triggers multiple hop condition explained next.

A window Ψ may perform multiple hops in order to improve the speed of the algorithm. There may be continuous regions of the Lidar space with no data at all. All such areas will be filled with 0 in the binary image space according to Equation (4.13). And, the pixel density (δ) for such areas will be 0 according to Equation (4.14). For successive hops of Ψ , if pixel density δ too successively turns out to be zero, then a probability of δ turning out to be zero ($p_{\delta \rightarrow 0}$) for the next hop is assumed. This probability is a function of a variable ‘confidence’ denoted by ζ , where $0 \leq \zeta \leq T_\zeta$ and T_ζ represents the threshold confidence. The value of ‘confidence’ increases or decreases according to the value of δ calculated with each hop of Ψ as,

$$p_{\delta \rightarrow 0} \propto \zeta \quad (4.16)$$

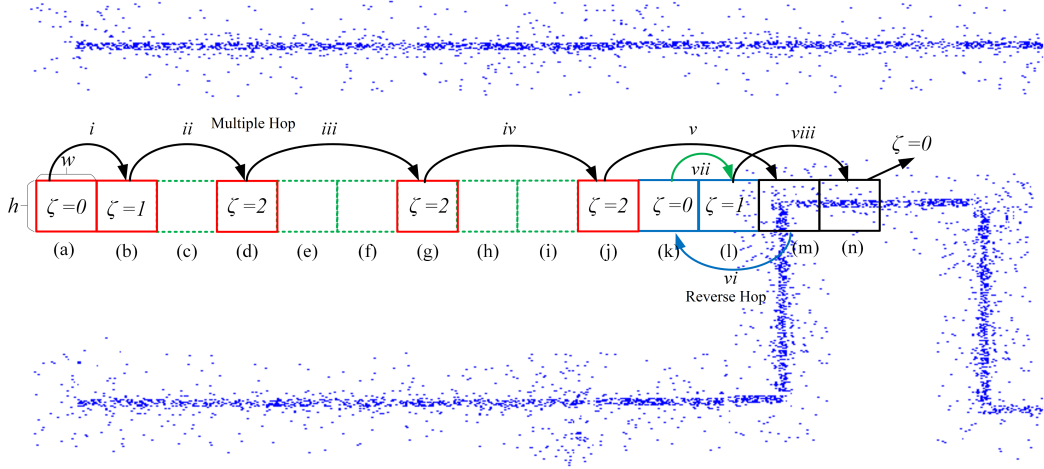


Fig. 4.17 The reverse-hopping trigger condition.

For continuous regions of $\delta = 0$, the window Ψ may perform multiple hops (i.e. skip the next hop and move on to the next to next region), as continuous scanning will result in unnecessary calculation of δ , which will be a waste of computational time.

Initially the value of ‘confidence’ ζ is set to 0. For the first time when window Ψ calculates $\delta = 0$, ζ is still set to 0. If, δ is calculated as zero for the consecutive hop, ζ is increased by 1. This process is repeated and the value of ζ is increased by 1 provided $\zeta < T_\zeta$. Once, the value of ζ becomes equal to the confidence threshold $T_{\beta_{eta}}$, ζ is saturated at the value T_ζ . Formally, the value of ζ is given by Equation (4.17).

$$\zeta = \begin{cases} 0, & \text{if } \delta = 0, 1^{st} \text{ scan} \\ \zeta_{old} + 1, & \text{if } \delta_{consecutive} = 0 \text{ and } \zeta_{old} < T_\zeta \\ T_\zeta, & \text{if } \delta_{consecutive} = 0 \text{ and } \zeta_{old} \geq T_\zeta \\ 0, & \text{if } \delta \neq 0 \end{cases} \quad (4.17)$$

Multiple hops of Ψ is performed for $\zeta \leq T_\zeta$. However, after performing multiple hops, if a condition of $\delta \neq 0$ is reached, it indicates that data might be in the proximity of the window Ψ and the skipped windows may contain actual data. Hence, ‘reverse-hop’ is performed and window Ψ goes back to the skipped locations. Specifically, Ψ reverse-hops by ζ_{old} times and the new ζ is again set to 0. Pixel density δ is calculated and forward hops is resumed. For $(\zeta_{old} - 1)^{th}$ forward hop, the value of ζ is set to 1, because pixel-density at the next window has already been calculated (since it is the window where reverse hop was performed), and the operation is resumed.

This is explained graphically in Figure 4.17. The value of ζ is set to 0 and confidence threshold T_ζ is set to 2, at the beginning of the algorithm. The window encounters $\delta = 0$ at window location (a) shown in Figure 4.17. For the first case, ζ is set to 0 according to Equation (4.17) and window hops to (b). Again, $\delta = 0$ is calculated and ζ is set to 1 according to Equation (4.17) ($\zeta \leftarrow \zeta_{old} + 1$, $\zeta_{old} = 0$). Value of $\zeta = 1$ indicates to hop over 1 window space and processing of window (c) is emitted. This hop is represented by arrow *ii*. For window (d) too, δ is found to be 0 and hopping confidence ζ is set to 2 and processing at two windows (e) and (f) are omitted. At window location (g), δ is zero but no increment of ζ is performed as threshold ($T_\zeta = 2$) has been reached and (h) and (i) are skipped. In similar fashion, hop is performed from (j) to (m). At window location (m), $\delta \neq 0$, and this triggers condition for reverse hopping. Value of ζ is set to 0 and window ‘reverse-hops’ to location (k) from (m). In other words, ζ_{old} times reverse hop is performed ($\zeta_{old} = 2$).

Listing 4.1 Hopping Window Noise Removing Algorithm

```

1: // Parameters:
2: //
3: //   FWIN_HEIGHT -> height of window
4: //   FWIN_WIDTH  -> width of window
5: //   THRESHOLD   -> pixel density threshold
6: //   P[W][H]    -> binary matrix of Lidar data
7: //
8: .
9: .
10: .
11: .
12: void FWNoiseRemove()
13: {
14:     int w; //top-left 'x' coordinate of window
15:     int h; //top-left 'y' coordinate of window
16:
17:     for(h=0; h < H; h+=FWIN_HEIGHT)
18:     {
19:         for(w=0; w < W; w+=FWIN_WIDTH)
20:         {
21:             FWcore(w,h); //call core for window
22:         }

```

```
23:   }
24: }
25:
26:
27: // Core
28: void FWcore(int w, int h)
29: {
30:     int c, r;      // col, row
31:     int sum = 0; // sum of all HIGH points
32:     int WIN_AREA = FWIN_HEIGHT * FWIN_WIDTH;
33:     boolean flag = false;
34:
35:     // scan window
36:     for(r=h;r<h+FWIN_HEIGHT&&!flag;r++)
37:     {
38:         for(c=w;c<w+FWIN_WIDTH&&!flag;c++)
39:         {
40:             if(P[r][c]==HIGH)
41:             {
42:                 sum++;
43:                 if(sum/WIN_AREA > THRESHOLD)
44:                 {
45:                     // break loop
46:                     flag = true;
47:                 }
48:             }
49:         }
50:     }
51:
52:     // if pixel density < THRESHOLD
53:     if(!flag)
54:     {
55:         // make all LOW in that window
56:         for(r=h;r<h+FWIN_HEIGHT;r++)
57:         {
58:             for(c=w;c<w+FWIN_WIDTH;c++)
59:             {
```

```

60:         R[r][c] = LOW;
61:     }
62: }
63: }
64: }

```

Notice that, for window location (l), ζ is set to 1, as location (l) is the ζ_{old}^{th} hop from previous location (j) ($\zeta_{old} = 2$). Thus, location (m) is skipped and window hops from location (l) to (n) and reprocessing of window (m) is prevented (location ‘m’ was already processed and was the location where ‘reverse-hop’ took place, so there is no need to process it again). The operation resumes henceforth from location (n) with $\zeta = 0$.

Multiple hop mechanism provides a way to speed up processing, while at the same time ‘reverse-hop’ mechanism prevents the omission of actual data processing which may occur by multiple hops. It is obvious that certain special data shapes may still cause omission of data processing by multiple hop mechanism especially if the hopping confidence (T_ζ) is set to a large value. Hence, a smaller value of T_ζ is good for the proposed algorithm.

Listing 4.1 shows the implementation of hopping window noise removal algorithm in Java language. Notice that, the algorithm showcases the core part of the algorithm and does not show the multiple hopping mechanism to preserve space. However, it can easily be incorporated in the algorithm.

4.4.2 Parallel implementation of the algorithm

The proposed algorithm is embarrassingly parallel. The binary image $P(i, j)$ where $0 \leq i < H$ and $0 \leq j < W$, can be divided into different subspace and each subspace can be processed ‘independently’ of the other thread/process for removing noise. Figure 4.18 shows binary image of Lidar data divided into four quadrants of size $\frac{W}{2} \times \frac{H}{2}$ denoted by Q1 to Q4. Each of the quadrants can independently be processed for noise removal. As shown in Figure 4.18, quadrant Q2 has no data while the other quadrants have Lidar data. Thus, the thread/process will perform multiple hops in quadrant Q2. Each thread/process will have its own ζ , however, all the threads/processes will use the same values for threshold confidence (T_ζ), frame window width (w) and height (h). The values of hopping window width (w) and height (h) are selected such that they perfectly divide the binary image $P(i, j)$ ’s width (W) and height (H), respectively. Similarly, while dividing the binary pixel image $P(i, j)$ into sub-spaces, hopping widow width

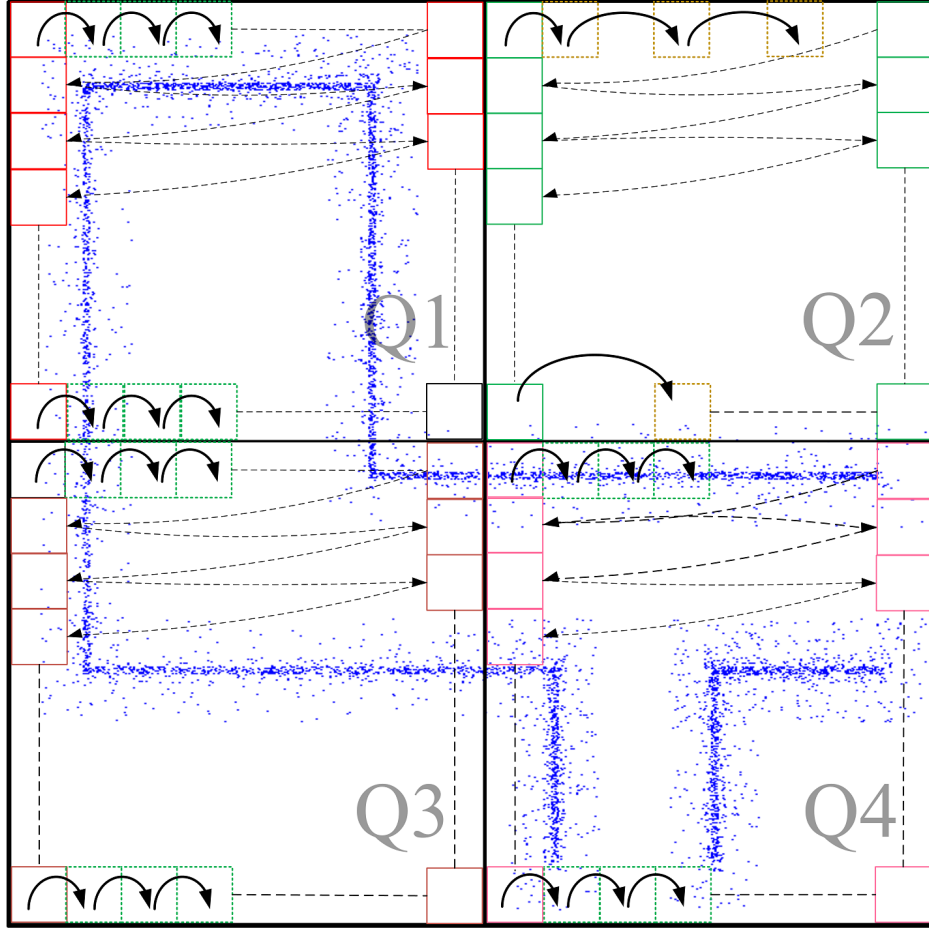
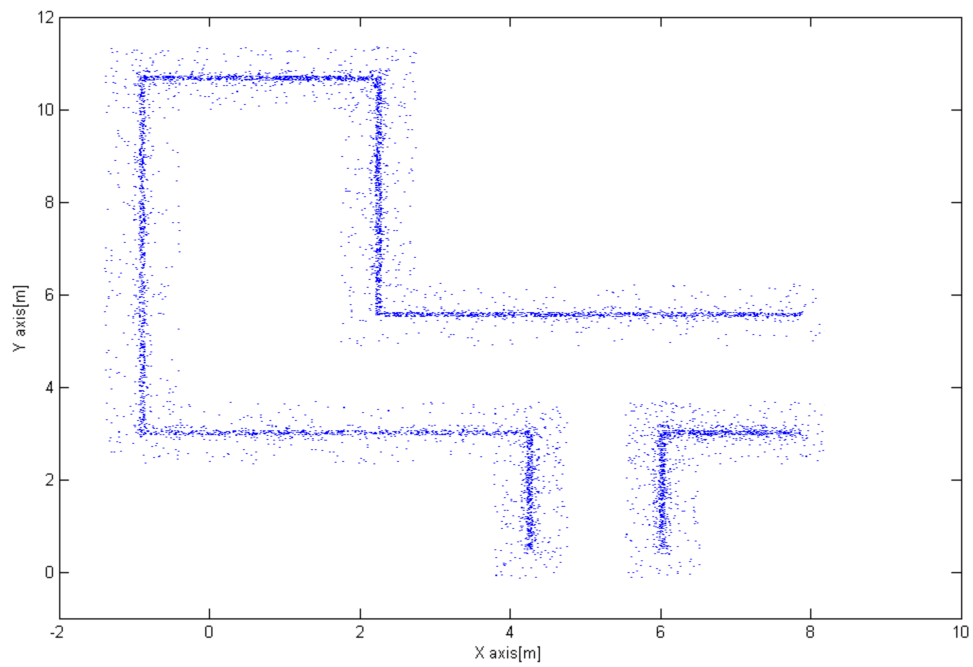


Fig. 4.18 Parallel implementation of hopping window noise removing algorithm implemented with 4 threads/processes. Each thread/process works on one of the quarters of the images from Q1 to Q4.

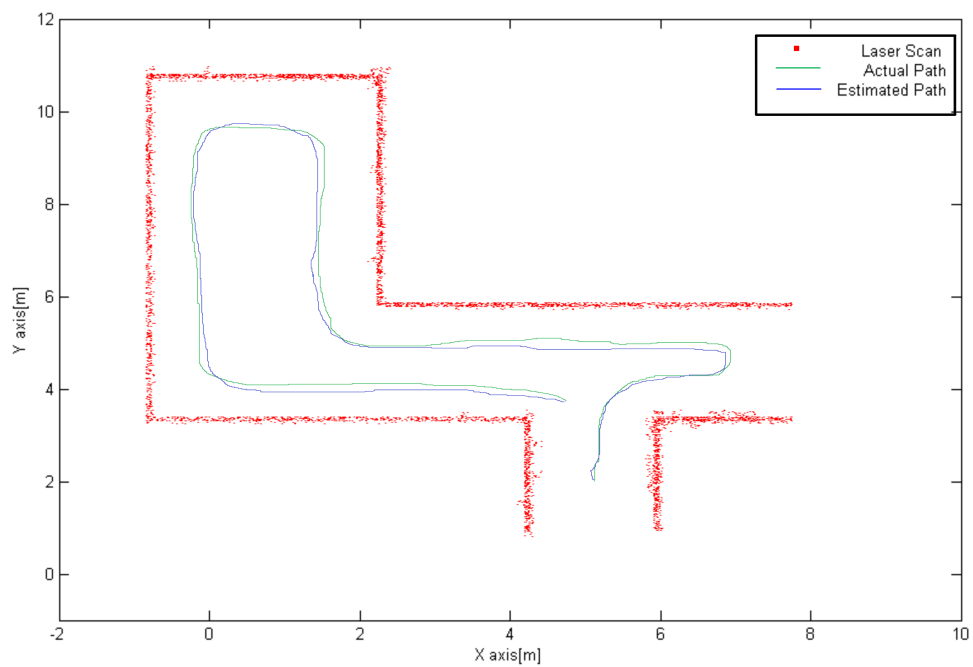
and height must be such that they perfectly divide the width ($W/2$) and height ($H/2$) of the subspace, respectively. Notice also that the proposed algorithm is thus very suitable for parallel implementation on GPU and other special multicore architectures [82].

4.4.3 Result

Fig.4.19a shows the binary image constructed from Lidar data. This image serves as an input of the proposed algorithm. Noise was randomly added to test our algorithm. The binary image was a square buffer of size $N \times N$ pixels, where $N=3600$ ($W=H=N$). The height (h) and width (w) of the hopping window was both set to 10. The pixel-density threshold (T_c) was set to 0.6. Figure 4.19b show the result after removing noise. The time required was calculated for noise removal by averaging the time of



(a) Binary image constructed from Lidar data. Noise was randomly added around Lidar data.



(b) Result of noise removal using hopping window algorithm.

Fig. 4.19 Results of noise reduction using Hopping window

100 iterations for Figure 4.19b. The average time required for noise removal of the test data shown in Figure 4.19a using serial execution was 34 milli seconds. A parallel implementation of the algorithm by using four threads and dividing the binary image into four quadrants as shown in Figure 4.18 and the processing was completed in 19 milliseconds. The algorithm is trained to determine the values of T_c , h and w before starting the algorithm for Hokuyo URG-04LX Lidar sensor. This initial training may be one of the limitations of the proposed algorithm. But, the algorithm is very fast and its implementation too is straightforward.

4.5 Clustering Algorithms

There are a number of clustering algorithms proposed for different applications [48, 105] and to discuss all of them in this thesis would not be feasible, however, in context with the thesis work two major clustering algorithms will be discussed in the following sections.

1. *k-means Clustering*
2. *Fuzzy c-means clustering*

4.5.1 k-means

k-means is a prototype-based, partial clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids. k-means defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically applied to objects in a continuous n -dimensional space. k-means is one of the oldest and most widely used clustering algorithms [48, 97].

Basic k-means Algorithm

The k-means clustering technique is simple, and we begin with a description of the basic algorithm. We first choose K initial centroids, where K is a user-specified parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same. The k-means algorithm is described by Algorithm 2 [97].

Figure 4.20 describes the operation of k-means, which shows how from three centroid points the final clusters are formed in subsequent iterations. Each subfigure shows (1) the centroids at the start of the iteration (2) assignment of points to those centroids.

Algorithm 2 Basic K-Means Algorithm

-
- 1: Select K points as initial centroids.
 - 2: **repeat**
 - 3: Form K clusters
 - 4: Recompute the centroid of each cluster.
 - 5: **until** Centroids do not change.
-

The centroids are shown by “o” symbol. Also each cluster has the same marker shape. Figure 4.20(a) represents the input data for the k-means. In Figure 4.20(b) points are assigned to the initial centroids. After the points are assigned to the centroid, the centroid is then updated. In the second step the points are assigned to the updated centroids, and the centroids are updated again. Figure 4.20(c), (d) and (e) shows the centroids moving to the bottom of the groups. Figure 4.20(e) shows the termination of the k-means algorithm since no more convergence occur, the centroids are assigned to the respective clusters. Figure 4.20(e), k-means reaches a state in which no points are shifting from one cluster to another, in other words, k-means always converge to a solution. Hence the centroids don’t change [97]. The following paragraphs describe each step of the k-means algorithm in detail.

Assigning Points to the Closest Centroid

A proximity measure is essential to assign point to the nearest centroid. Euclidean distances (L_2) are often used for such data points. Since k-means algorithm repeatedly calculates similarity of each point to each centroid, the similarity measures used for k-means are relatively simple.

Data in Euclidean Space

The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids;

Table 4.2 Table of Notations for k-means algorithm

Symbol	Description
x	An object
C_i	The i^{th} cluster.
c_i	The centroid of cluster C_i
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

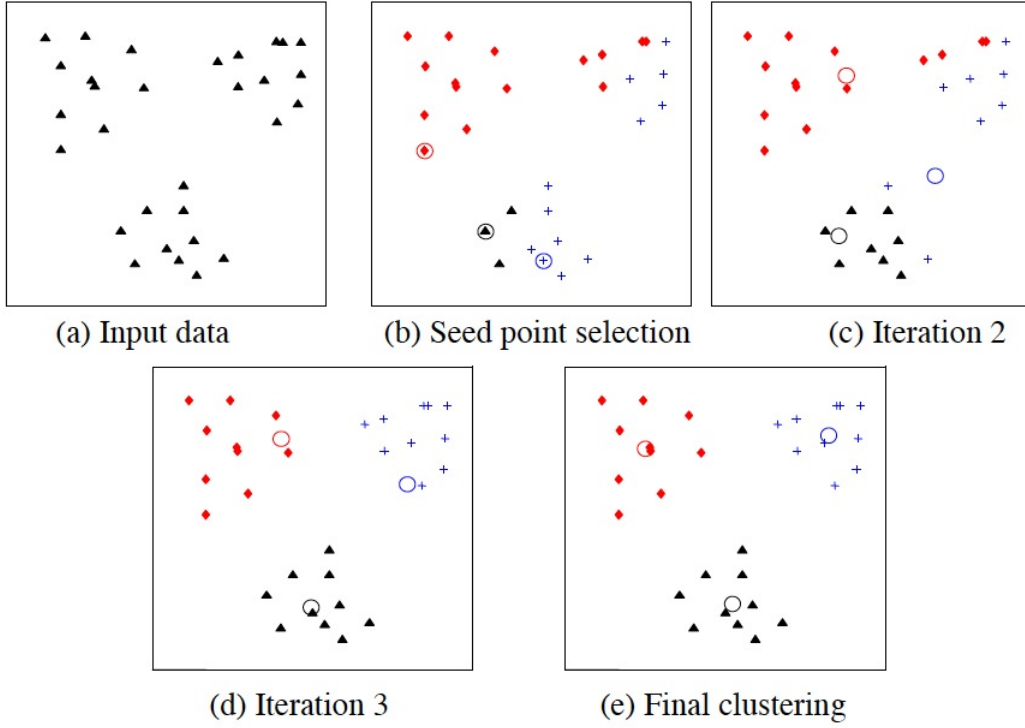


Fig. 4.20 Illustration of k-means algorithm. (a) 2D input data with three clusters; (b) three seed points selected as cluster centers and initial assignment of the data points to clusters; (c) and (d) intermediate iterations updating cluster labels and their centers; (e) final clustering obtained by k-means algorithm at convergence [48].

e.g., minimize the squared distance of each point to its closest centroid. Consider data whose proximity measure is Euclidean distance. For the objective function, which measures the quality of a clustering, we use the sum of squared error (**SSE**). In other words, we calculate the error of each data point, i.e. its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of k-means, we prefer the one with the smallest squared error since this means that the centroids of this clustering are a better representation of the points in their cluster. The SSE [48, 97] is defined as

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}(c_i, x)^2 \quad (4.18)$$

where 'dist' is the standard Euclidean distance (L_2) between two objects in Euclidean space. Given these assumptions, it can be shown that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in Table 4.2, the centroid (mean) of

the i^{th} cluster is defined by Equation (4.19),

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x \quad (4.19)$$

Steps 3 and 4 of the k-means algorithm directly attempt to minimize the SSE or the objective function. Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids. Step 4 recomputes the centroids so as to further minimize the SSE.

Derivation of k-means as an Algorithm to Minimize the SSE

In this section the mathematical derivation [97] of the k-mean algorithm when the proximity function as a Euclidean distance and the objective is to minimize the SSE. The derivation explains how to best update a cluster centroid to minimize the cluster SSE. Rewriting Equation (4.18) specialized in one dimensional data.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}(c_i - x)^2 \quad (4.20)$$

Using notations from Table 4.2, we can solve for the k^{th} centroid c_k , which minimizes Eq. (2.3) by differentiating the SSE, setting it equal to 0, and solving below.

$$\begin{aligned} \frac{\partial}{\partial c_k} SSE &= \frac{\partial}{\partial c_k} \sum_{i=1}^K \sum_{x \in C_i} (c_i - x)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} \frac{\partial}{\partial c_k} (c_i - x)^2 \\ &= \sum_{x \in C_k} 2 \times (c_k - x_k) = 0 \\ \sum_{x \in C_k} 2 \times (c_k - x_k) &= 0 \Rightarrow m_k c_k = \sum_{x \in C_k} x_k = c_k = \frac{1}{m_k} \sum_{x \in C_k} x_k \end{aligned} \quad (4.21)$$

Thus, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

Time and Space Complexity of k-means algorithm

Since only the data points and centroids are stored k-means has a moderate storage requirement. The storage required [48, 97] is $O((m + K)n)$, where m is the number of points and n is the number of attributes. The time requirements [97] for k-means are linear in the number of data points. The time required is $O(I \times K \times m \times n)$, where I

is the number of iterations required for convergence. Thus, k-means is linear in the number of points m , and is efficient and simple provided that number of clusters K , is significantly less than m .

4.5.2 Fuzzy c-means Clustering

The Fuzzy c -Means Clustering or FCM [10] is an extension of the k-Means clustering. FCM in many ways is similar to the k-Means clustering except that every data object in the data set is given membership by weight (generally 0 to 1), and each data object belong to every cluster with a weightage (0 for doesn't belong at all and 1 for absolutely belong). Thus every point in the data set belongs to each and every cluster with a probability (0 to 1). The objective function that FCM optimizes is

$$J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m \|x_i - c_j\|^2, \quad 1 \leq m \leq \infty \quad (4.22)$$

where $m \in \mathbb{R}$ and $m \geq 1$, $u_{i,j}$ is the degree membership of x_i in the cluster j , x_i is the i^{th} of the d -dimensional measured data, c_j is the d -dimensional center of the cluster, and $\|\cdot\|$ is any norm expressing the similarity between any measured data and the center.

Fuzzy partitioning is carried out by iterative optimization of the objective function J_m , with update membership u_{ij} and cluster centers c_j by

$$u_{ij} = \frac{1}{\sum_{k=1}^C \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}, \quad c_j = \frac{\sum_{i=1}^N u_{ij}^m x_i}{\sum_{i=1}^N u_{ij}^m} \quad (4.23)$$

the iteration stops when $\max_{ij} \left\{ |u_{ij}^{k+1} - u_{ij}^k| \right\} < \varepsilon$ where ε is a termination criterion between 0 and 1, whereas, k are the iteration steps. The process always converges to a local minimum or J_m .

4.5.3 FCM Algorithm

- **Step 1:** Initialize $U = [u_{ij}]$ matrix, $U^{(0)}$.
- **Step 2:** At k -step: calculate the centers vectors $C^{(k)} = [c_{ij}]$ with $U^{(k)}$

$$c_j = \frac{\sum_{i=1}^N u_{ij}^m x_i}{\sum_{i=1}^N u_{ij}^m}$$

- **Step 3:** Update $U^{(k)}, U^{(k+1)}$

$$u_{ij} = \frac{1}{\sum_{k=1}^C \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

- **Step 4:** if $\max_{ij} \{|u_{ij}^{k+1} - u_{ij}^k|\} < \varepsilon$ then STOP; otherwise return to Step 2.

4.5.4 Results - k-means and Fuzzy c-means

After removing the noise, clustering is applied on the noise-free LRF data. Clustering can be performed on real-time LRF data after each *step* in *Online Mode* or after a predefined N_{smax} *steps* in *Offline Mode*. The choice of the mode of operation of the robot will also determine the number of clusters performed at each *step*. For *Online Mode*, the number of clusters (and hence the centroid) formed at each step is one, while for *Offline Mode* it can be greater than one (≤ 4 in our test environment). Either *k*-means or fuzzy *c*-means clustering is applied.

Figure 4.21a and Figure 4.21b shows the result of applying *k*-means clustering and *c*-means clustering on noise-free LRF data with 52 clusters, respectively. The different colours applied to the LRF data clearly show the different clusters and have no other significance. The centroids obtained are also shown as circles in the case of *k*-means clustering and rectangles in the case of *c*-means clustering in Figure 4.21a and Figure 4.21b, respectively. Later, these centroids are joined to form the map.

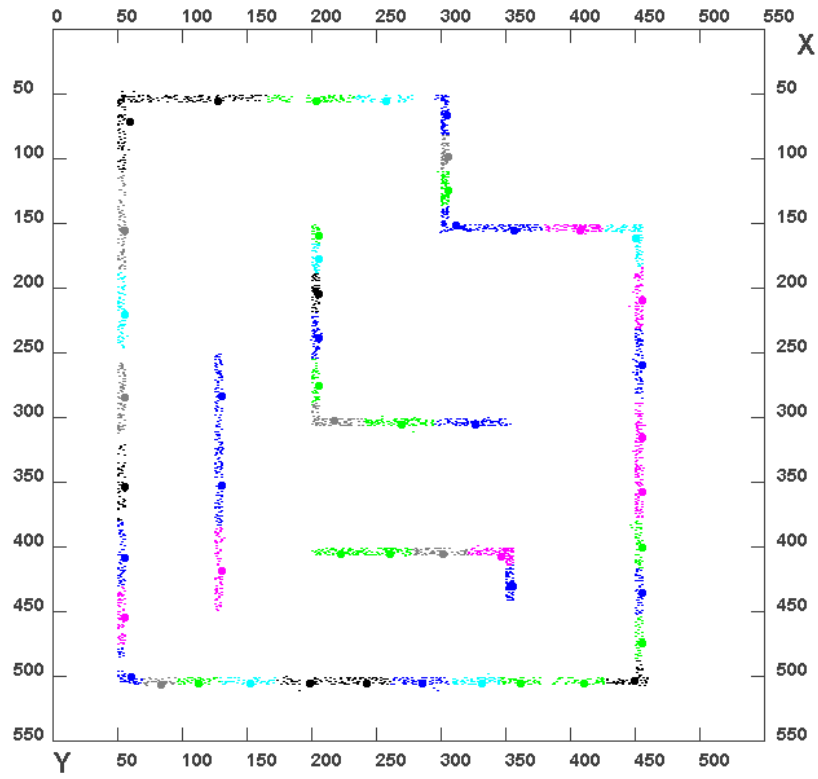
4.5.5 Generating Straight-line Maps

If the centroids shown in Figure 4.21a and 4.21b are joined, exact straight-line maps may not be obtained as the centroids do not lie on the same slope [79]. In some cases, it may be desirable to obtain straight-line maps. This can be solved by applying SVD to the centroids or by taking a Hough transform of the centroids. A brief description of both approaches is given in the following sections.

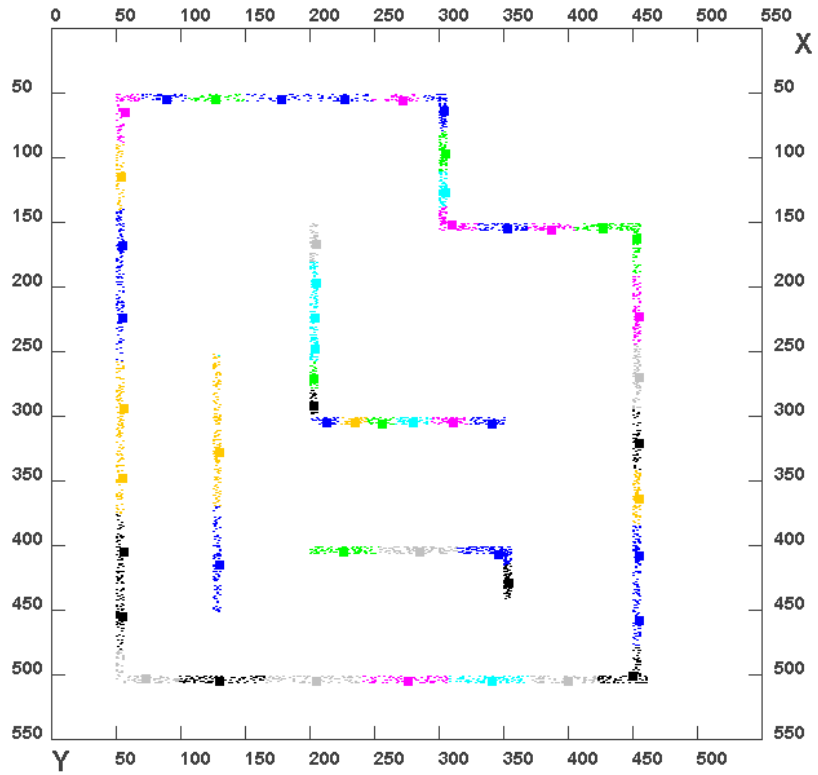
Applying SVD to the Centroids of the Clusters

A straight line can be obtained by applying SVD [66, 78, 83] on the matrix A which is composed of the centroids $(c_1, c_2 \dots c_n)$ obtained after the clustering of the LRF data for a given θ :

$$A = \begin{bmatrix} c_1 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} \quad (4.24)$$



(a) k-means result with 52 clusters.



(b) Fuzzy c-means result with 52 clusters.

Fig. 4.21 Results of k-means and fuzzy c-means clustering on noise free data.

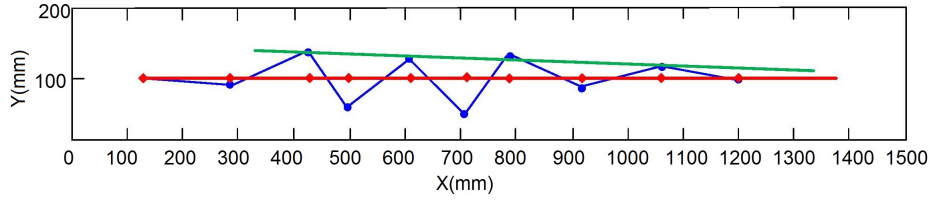


Fig. 4.22 Straight-line generation from centroids using SVD and Hough transform. The blue points represent the centroids, the red line represents the SVD line, while the green line represents the line obtained from the Hough transform.

The SVD of a matrix A is expressed as:

$$A = USV^T \quad (4.25)$$

where U and V^T are orthonormal matrices consisting of the left and right singular vectors, and S is the diagonal matrix which is formed by the singular values arranged in decreasing order of the on-diagonal elements as follows:

$$U = [u_1 \cdots u_n] \quad (4.26)$$

$$S = \begin{bmatrix} \text{diag}(\sigma_1 \sigma_2 \cdots \sigma_n) \\ 0 \end{bmatrix} \quad (\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n) \quad (4.27)$$

$$V = [v_1 \cdots v_n] \quad (4.28)$$

The matrices U and V satisfy $U^T U = I$ and $V^T V = I$, where the matrix I represents the identity matrix.

The SVD calculations consist of evaluating eigenvalues and eigenvectors of AA^T and $A^T A$. The eigenvectors of $A^T A$ make up the columns of V and the eigenvectors of AA^T make up the columns of U . Moreover, the singular values in S are square roots of the eigenvalues from AA^T or $A^T A$.

After computing the SVD of the centroids, the minimum (near-zero) singular values (σ) of the matrix S are ignored and set to zero. This is similar to the technique used in principal component analysis. The S_{new} with the smaller singular values ignored is then used to recalculate the matrix A_{new} consisting of points on the regression line as:

$$A_{new} = US_{new}V^T \quad (4.29)$$

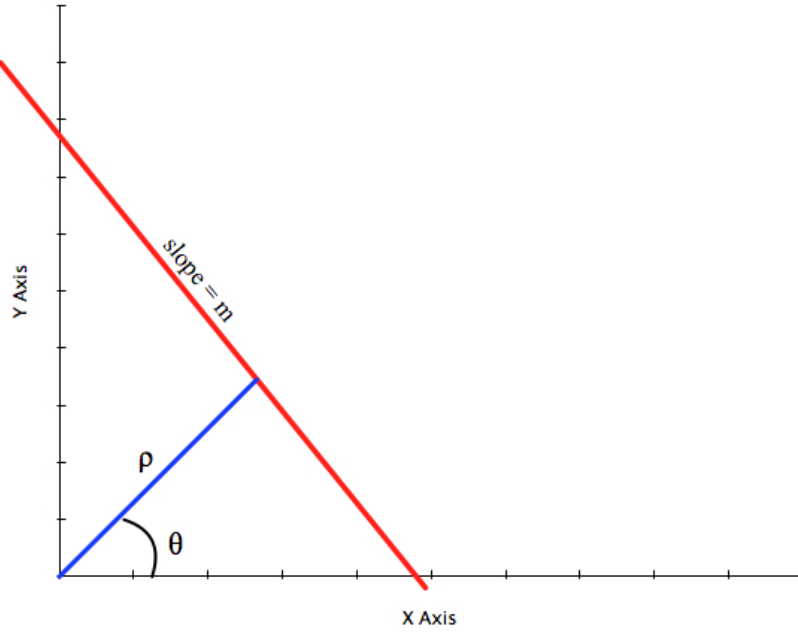


Fig. 4.23 Line representation in ρ and θ parameter.

Applying a Hough Transform to the Centroids of the Clusters

A Hough transform is a technique for line extraction from binary images and its details are available in [5, 23]. The motivating idea behind the the Hough transform technique for line detection is that each input measurement (centroids in this case) indicates its contribution to a globally consistent solution. In the spatial domain, the straight line can be expressed as:

$$y = mx + c, \quad (4.30)$$

The equation of the line in the Hough space (ρ, θ) can be expressed as (Figure 4.23):

$$\rho = x \cos \theta + y \sin \theta. \quad (4.31)$$

The Hough transform algorithm is applied on the centroids obtained from k-means or fuzzy c-means clustering. After calculating the highest votes in the (ρ, θ) space, an inverse Hough transform is applied which gives the line passing through the maximum number of centroids. The Hough transform and inverse Hough transform algorithm applied on the centroids is described in Algorithm 3.

Figure 4.22 shows the results of straight-line generation using SVD and the Hough transform on the centroids. The blue points represent the centroids of clusters obtained

Algorithm 3 Straight-line generation from centroids using a Hough transform.

```

1:  $P_i \leftarrow \{p_1, p_2, p_3, \dots, p_n\}$  /* centroids */
2:  $V(\rho, \theta) \leftarrow 0$ ; /* clear vote - accumulator */
3:  $Max\_ \theta$ ; /* max  $\theta$  */
4:  $W$ ; /* max  $x$  size */
5: /* Convert points  $P(x_i, y_i)$  to  $V(\rho_i, \theta_i)$  */
6: for  $\forall x_i \in P_i$  do
7:   for  $\forall y_i \in P_i$  do
8:     if  $P(x_i, y_i) == \text{Bright}$  then
9:       /* Eval :  $P(x_i, y_i) \rightarrow V(\rho_i, \theta_i)$  */
10:      for  $0 \leq \theta \leq Max\_ \theta$  do
11:         $\rho \leftarrow x_i \times \cos(\theta) + y_i \times \sin(\theta)$ 
12:        /* increment vote - accumulator */
13:         $V(\rho_i, \theta_i) \rightarrow V(\rho_i, \theta_i) + 1$ ;
14:      end for
15:    end if
16:  end for
17: end for
18: /* Find Max  $V(\rho_i, \theta_i)$  */
19:  $T_{\rho, \theta} \leftarrow 0$ 
20: for  $\forall \rho_i \in V_i$  do
21:   for  $\forall \theta_i \in V_i$  do
22:     if  $T_{\rho, \theta} < V_{\rho_i, \theta_i}$  then
23:        $T_{\rho, \theta} \leftarrow V_{\rho_i, \theta_i}$ ;
24:     end if
25:   end for
26: end for
27: /* Extract line from  $T_{\rho, \theta}$  */
28: for  $\forall x_i \leq W$  do
29:    $\rho \leftarrow T_\rho$ 
30:    $\theta \leftarrow T_\theta$ 
31:    $y_i \leftarrow \frac{(\rho - x_i \times \cos(\theta))}{\sin(\theta)}$ 
32:    $Line_{x_i, y_i} \leftarrow \text{Bright}$ 
33: end for
34: Draw  $Line_{x_i, y_i}$ 

```

after the noise reduction and k-means clustering of a wide band of LRF data. The red line represents the line obtained after applying SVD. The green line represents the line obtained by using the Hough transform. It is clear from Figure 4.22 that SVD generates the regression line, whereas the Hough transform generates a line which passes through the maximum number of centroids. Since the number of clusters n_c is significantly less than the actual number of data points N (i.e., $n_c \ll N$), the Hough transform and the SVD are relatively inexpensive in terms of computation and memory in obtaining a straight line from the centroids.

Thus, the process of map generation in the spatial domain involves the following steps:

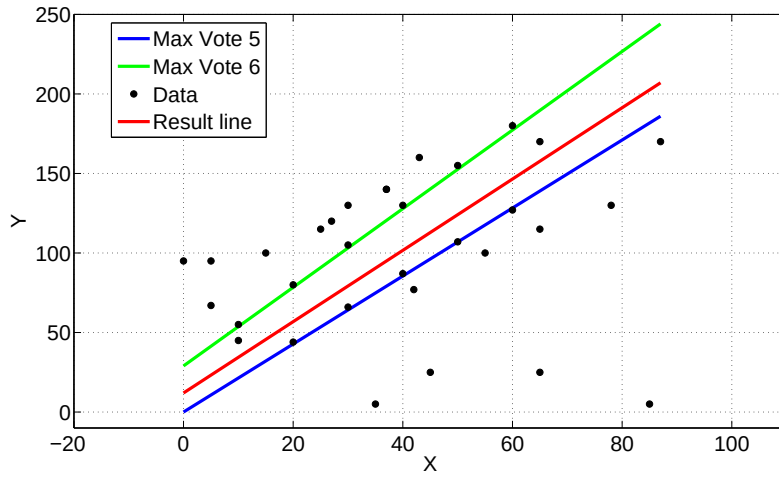
1. Gather LRF data in *Online*, *Offline* or *Mixed* Mode and segment data.
2. Remove noise by DBSCAN or mathematical morphological kernels.
3. Apply k-means or fuzzy c-means clustering.
4. Join centroids obtained after clustering.
5. Obtain straight-line maps by applying SVD or Hough transform to centroids.

The accuracy of the line maps obtained by joining the centroids around corners is affected by the number of clusters. One way to solve this problem is to increase the number of clusters. Furthermore, to overcome the extraction of the corners, other techniques like the split and merge [13, 71, 73] and IEPF [24] algorithms discussed earlier can be used.

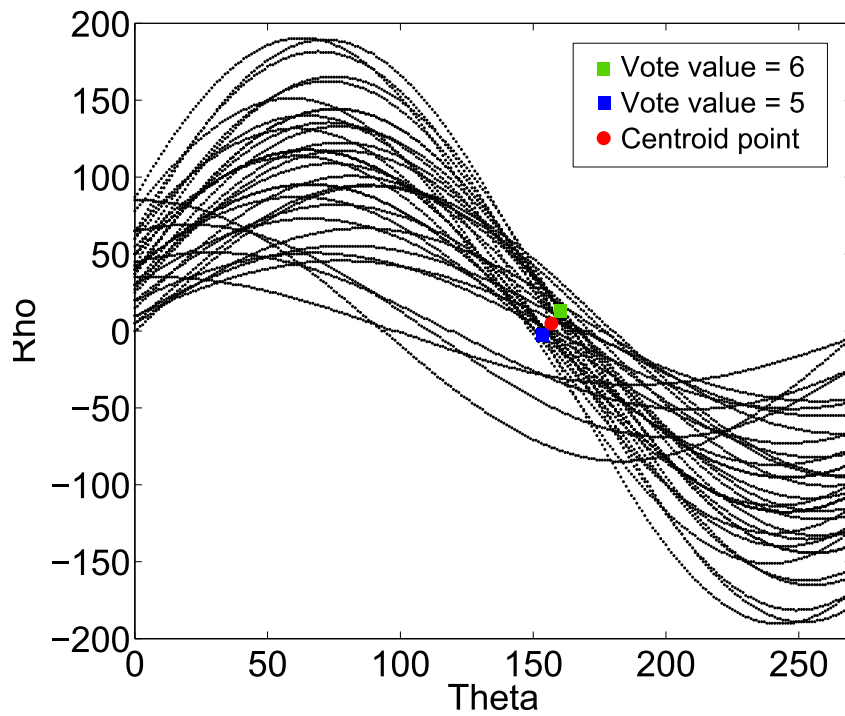
4.6 Clustering in the Hough Domain (Accumulator Space)

The previous section discussed clustering in the spatial (i.e., xy) domain, in which clustering (k-means or fuzzy c-means) was directly applied to the LRF data after removing noise. It is necessary to first remove noise and then apply clustering because clustering is greatly affected by noise [56]. However, we can overcome this limitation by clustering in the Hough (ρ, θ) domain.

The Hough transform generates a straight line by looking into each LRF data point and, if there is enough evidence of an edge on that point, the Hough transform calculates the parameters for that line and increments the corresponding accumulator

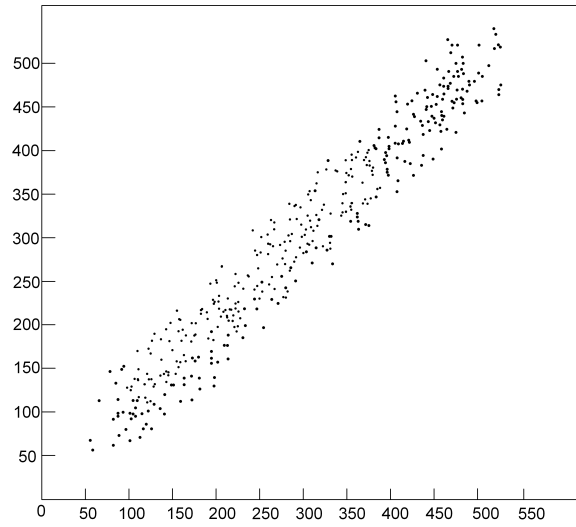


(a) Line extraction by clustering in the Hough domain. The green line and the blue line correspond to vote values of '6' and '5', respectively. The red line is drawn against the centroid point. ($\delta = 1$)

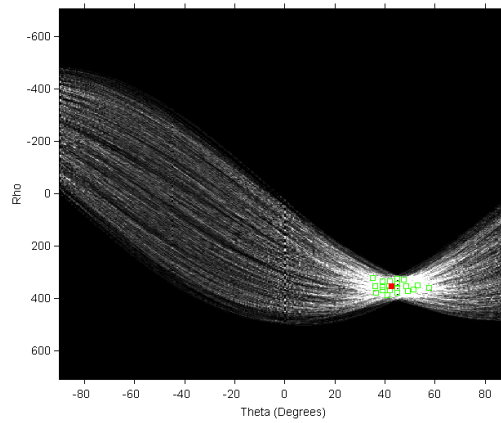


(b) The Hough accumulator space showing a vote value of '6' in green and a vote value of '5'. Their centroid is indicated in red.

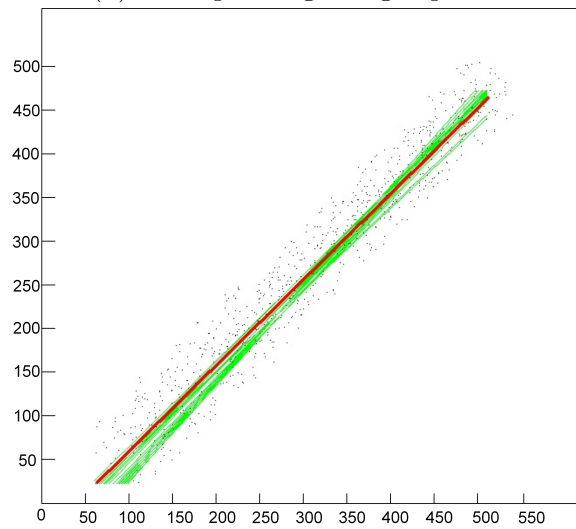
Fig. 4.24 Map generated using clustering in the spatial domain.



(a) Sample laser sensor scan data,



(b) corresponding Hough space.



(c) single straight line (red)

Fig. 4.25 Line generated after clustering the Hough Space parameters. The red line is the optimum line obtained by clustering in the Hough domain.

vote value [5, 23]. By finding the highest number of votes, or by looking for the local maxima in the accumulator space, the most-likely lines are extracted. However, with this method, votes which are even slightly lower than the threshold or the local maxima are ignored or else not taken into consideration while extracting the line. This is not good, especially when the data contains a lot of noise. Moreover, for long-range measurements, the noise increases for long-range sensors.

A Hough transform applied on a set of collinear points will converge at a single peak representing the highest number of votes in the accumulator space. Applying an inverse Hough transform on the peak point will generate a single straight line. However, in the real world, when the robot is moving, the LRF data is not collinear. An example of actual LRF data is shown in Figure 4.25a. Applying a Hough transform on this LRF data will generate a Hough space, as shown in Figure 4.25b, in which the (ρ, θ) curves do not intersect at a single point but rather at many intersecting points. Each intersecting point in the Hough space will correspond to a straight line which passes through one or more of the LRF data points. For the highest number of votes in the accumulator space, 13 green lines were produced, as shown in Figure 4.25c.

Researchers have proposed many solutions [31, 41, 49, 101] for obtaining the most optimum parameter value by which a single straight line can be produced. Most of them are proposed for finding straight lines in real images for boundary or feature extraction. For the purpose of obtaining a single line for the LRF data, this thesis proposes a new algorithm using the clustering of Hough space parameters.

To obtain a single straight line from the LRF data, we extend the voting procedure in the accumulator space to clustering[84]. After getting the maximum vote value from the Hough space parameters (ρ, θ) , clustering is applied to the accumulator votes which lie within a threshold in order to evaluate and compute a centroid. Generally, the highest vote value from the (ρ, θ) accumulator space is selected and a corresponding single line is obtained from that value. However, there are two problems with this approach. Firstly, there can be multiple points in the (ρ, θ) accumulator space with same highest-vote value. Each highest-vote value of (ρ, θ) will generate its own line, which will result in multiple line extractions. To get a single line for the purpose of building meaningful maps, the highest voted points are clustered and a centroid is obtained. The single line corresponding to the centroid point (ρ, θ) will receive contributions from all the highest voted points. Secondly, in a noise-free environment, the centroid of only the highest voted rho-theta ($Vmax_{rt}$) point can be calculated and a single line can be extracted to this point. However, for (ρ, θ) points whose vote

Algorithm 4 Mapping using clustering in the Hough domain.

```

     $P_i \leftarrow \{p_1, p_2, p_3, \dots, p_n\}$  /* datapoints */ 30:   in range  $\delta$  */
    /
    2:  $V(\rho, \theta) \leftarrow 0$ ; /* clear vote - accumulator */ 32:    $T_{\rho, \theta} \leftarrow 0$ 
     $Max\_ \theta$ ; /* max  $\theta$  */ 34:   for  $\forall \rho_i \in V_i$  do
    4:  $W$ ; /* max  $x$  size */;  $C$ ; /* cluster */ 36:       for  $\forall \theta_i \in V_i$  do
     $k$ ; /* points to be clustered */ 38:           if  $T_{\rho, \theta} - \delta \leq V_{\rho_i, \theta_i}$  then
    /,  $\delta$ ; /* min difference range */ 38:                $C.join(V_{\rho_i, \theta_i})$ ;
    6: for each  $S^n$  do 38:                    $k \leftarrow k + 1$ 
    /* Convert points  $P(x_i, y_i)$  to  $V(\rho_i, \theta_i)$  */ 40:               end if
    / 40:           end for
    8:   for  $\forall x_i \in P_i$  do 42:   end for
    for  $\forall y_i \in P_i$  do 42:   /* Randomly select a centroid */
    10:       if  $P(x_i, y_i) == \text{Bright}$  then 42:    $K \leftarrow \{C_1, C_2, C_3, \dots, C_k\} : K \in C$ ;
    /* Eval :  $P(x_i, y_i) \rightarrow V(\rho_i, \theta_i)$  */ 44:    $flag\_converge \leftarrow FALSE$ ;
    /; 46:   while ! $flag\_converge$  do
    12:       for  $0 \leq \theta \leq Max\_ \theta$  do 46:        $recompute\_centroid(P_k)$ ;
     $\rho \leftarrow x_i \times \cos(\theta) + y_i \times \sin(\theta)$  48:       if Centroids  $P_K == \text{converged}$ 
    14:       / * increment vote - accumulator */ 48:       then
     $V(\rho_i, \theta_i) \rightarrow V(\rho_i, \theta_i) + 1$ ; 50:        $flag\_converge \leftarrow TRUE$ ;
    16:       end for 50:       end if
    end if 52:   end while
    18:   end for 52:    $C_\rho \leftarrow C_{\rho, \theta}$ 
    end for 52:    $C_\theta \leftarrow C_{\rho, \theta}$ 
    20: / * Find Max  $V(\rho_i, \theta_i)$  */ 54:   /* Extract line from  $C_{\rho, \theta}$  */
     $T_{\rho, \theta} \leftarrow 0$  54:   for  $\forall x_i \leq \text{width}$  do
    22:   for  $\forall \rho_i \in V_i$  do 56:        $\rho \leftarrow C_\rho$ 
    for  $\forall \theta_i \in V_i$  do 56:        $\theta \leftarrow C_\theta$ 
    24:       if  $T_{\rho, \theta} < V_{\rho_i, \theta_i}$  then 58:        $y_i \leftarrow \frac{(\rho - x_i \times \cos(\theta))}{\sin(\theta)}$ 
     $T_{\rho, \theta} \leftarrow V_{\rho_i, \theta_i}$ ; 58:        $Line_{x_i, y_i} \leftarrow \text{Bright}$ 
    26:       end if 60:   end for
    end if 60:   Draw  $Line_{x_i, y_i}$ 
    28:   end for 60:   end for
    /* Generate clusters of votes  $V(\rho_i, \theta_i)$ 

```

value is slightly less than the highest vote (ex. $V_{max_{rt}} - 1$), these do not contribute to the final line and they are ignored when the environment is noisy. These slightly low voted points can also be taken into account while calculating the centroid, and the generated line will have contributions not only from the highest voted points but also from the slightly less voted points. This is shown in Figure 4.24. Figure 4.24a shows a sample lidar data scan with noise. Figure 4.24b shows the corresponding Hough voting space. The green line in Figure 4.24a is the line passing through the maximum number of points (corresponding to the highest vote value '6', shown as a green dot in Figure 4.24b), and in a noise free environment this will be the desired line. The blue line is the line passing through slightly fewer points (corresponding to a vote value of '5', shown as a blue dot in Figure 4.24b). In a noisy environment, this line should also contribute to the final result line. The red line represents the passing line corresponding to the centroid of the highest voted (ρ, θ) point with six votes and the slightly less voted point with five votes, shown as a red dot in Figure 4.24b.

Points with low votes generally correspond to noise and are not taken into account while clustering, and thus the effect of noise is greatly reduced particularly in the case of the LRF data. By taking the inverse Hough transform of the centroid, the most optimum line is obtained, which has contributions from the highest voted points. For the Hough space shown in Figure 4.25b, the most optimum line obtained after the inverse Hough transform is shown in Figure 4.25c as a red line. Clustering can be done using k-means or fuzzy c-means clustering.

The algorithm is described in Algorithm 4 and explains the clustering of the Hough parameters using k-means clustering. For each segment of data S^n obtained after applying segmentation described in 4.2.2, the algorithm begins by taking the Hough transform of the LRF data. Next, the highest vote value (denoted by $T_{\rho, \theta}$) in the (ρ, θ) domain is found. In the next step, all the values of (ρ, θ) denoted by $V_{\rho, \theta}$ whose values are within the range δ (i.e., $T_{\rho, \theta} - \delta \leq V_{\rho, \theta}$) are extracted. Later, clustering is applied on the set of all such points and a centroid is obtained. Finally, the inverse Hough transform of the centroid point is taken to generate a line. The value of δ is generally set to '0', '1' or '2'. The higher values of δ will enable the clustering of (ρ, θ) points with lesser votes, which is not desirable. In the case of multiple lines, the LRF points corresponding to each line will be segmented into respective segments S^n described in Section 4.2.2, where n is the number of lines. The Hough transform, clustering and inverse Hough transform are applied separately in each segment and lines are

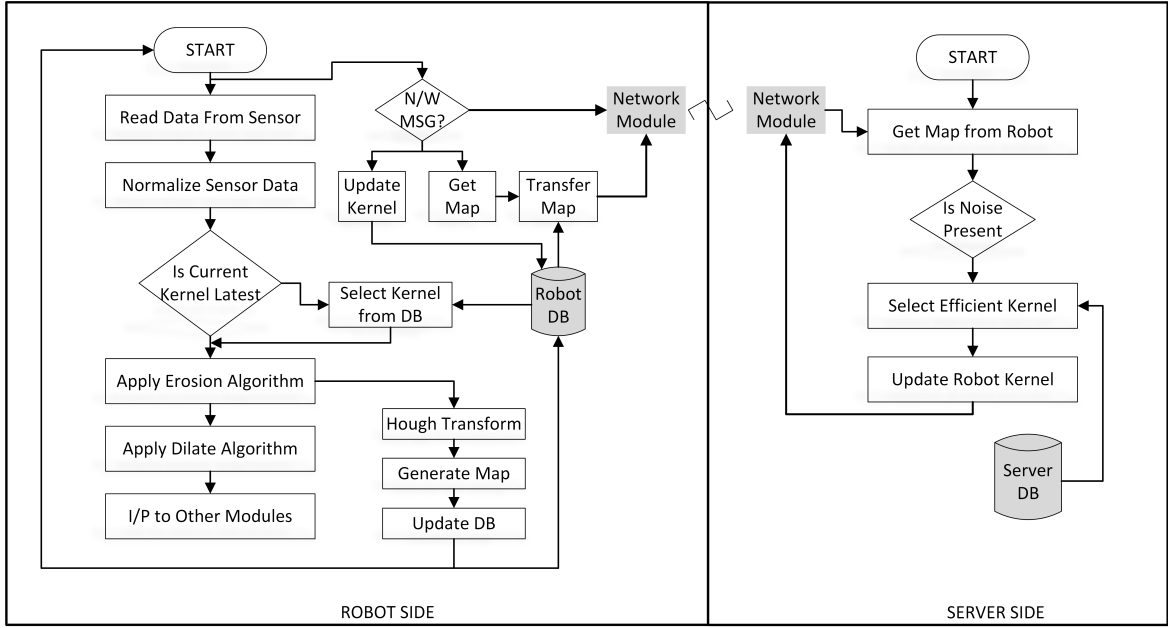


Fig. 4.26 Framework for robot map building showing the robot side and the server side. The message is exchanged in xml format between robot and server.

extracted. If p_i^n denotes the set of i vote points corresponding to the n^{th} segment S , then clustering is separately performed on the points in each segment, given by:

$$p_i^n = \{V_{\rho, \theta}\} \quad (\text{for each } S^n \text{ separately}), \quad (4.32)$$

$$\forall \rho, \theta \quad \text{where} \quad T_{\rho, \theta} - \delta \leq V_{\rho, \theta}$$

For k -means clustering in the Hough domain, Equation (4.18) is used. In the case of Figure 4.24, Equation (4.32) represents two (ρ, θ) points with vote values of '6' and '5', shown as green and blue dots in Figure 4.24b, respectively ($\delta = 1$). Notice that the centroid obtained after clustering may or may not belong to the set of points denoted by Equation (4.32). For example, in the case of Figure 4.24, the centroid does not belong to the set of points given by Equation (4.32).

4.7 Framework

This section describes the framework for robot map building. The robot has a database, denoted by 'Robot DB' in Figure 4.26. This saves the kernels (Erode/Dilate, DBSCAN, k-means, fuzzy c-means, etc.). It also saves the temporary maps generated by the robot. The kernels can be updated by the server and the updated kernels

are also saved in the database. The robot reads data from the LRF sensor which is then normalized. An appropriate kernel is selected from the database to remove noise. This can either be DBSCAN or Erode/Dilate or both, depending upon the instruction from the server. Before actually applying the kernel to the normalized data, a check is performed to see if the kernel is the most updated kernel. The necessary operation is performed (ex. Erosion and Dilation) and the resultant data is the input to the clustering algorithm, the Hough transform algorithm or other data-processing modules. The temporary map which is generated is saved in the robot database. The network module of the robot continuously polls to check if the temporary map has been updated and stored, and then transfers this map to the server for real-time display. This real-time display of the map at the server enables the tuning of the parameters of various kernels (e.g., select appropriate kernels or change the number of clusters). The server also has a database to store updated kernels.

The same network module is used to update the parameters or kernels of the robot which are stored in the robot's database. Since the robot continuously checks the database for updated parameters, they can be tuned from the server.

The message to update/tune the parameters kernel is carried out via an xml file which is parsed using an xml-parser by the robot to select an appropriate kernel and corresponding parameter values (ex. ϵ , minimum points in the case of DBSCAN, number of clusters). Listing 4.2 shows an example of such a message passed from the server to the robot. The xml message contains the instruction to select the DBSCAN noise removal kernel with 15 minimum points and an ϵ value of '5'. It also instructs the robot to remove noise using the PLUS $\times$ 4 kernel. The instructed mode is *OnlineMode*. The method of clustering is the k-means algorithm with two clusters. The straight line is to be generated by taking the Hough transform of the centroids.

Listing 4.2 Message from server to robot

```

1: <?xml version="1.0"?>
2: <xml>
3:   <message>
4:     <mode>
5:       online
6:     </mode>
7:     <noise-removal>
8:       <dbscan>
9:         <minpts>

```

```

10:      15
11:      </minpts>
12:      <epsilon>
13:      5
14:      </epsilon>
15:    </dbscan>
16:    <erosion>
17:      <kernel-name>
18:        PLUSx4
19:      </kernel-name>
20:    </erosion>
21:  </noise-removal>
22:  <clustering>
23:    <kmeans>
24:      <cluster-no>
25:        2
26:      </cluster-no>
27:    </kmeans>
28:  </clustering>
29:  <straight-line-method>
30:    Hough
31:  </straight-line-method>
32: </message>
33: </xml>

```

4.8 Experimental Results

Table 4.3 Percentage error of the map area for clustering in the spatial domain for an artificial environment as shown in Figure 4.11.

No. of Cluster	50 clusters			60 clusters			70 clusters		
Method	CJ	HT	SVD	CJ	HT	SVD	CJ	HT	SVD
<i>k-means</i>	2.917	2.850	2.813	2.571	2.453	2.438	2.176	2.163	2.138
<i>FCM</i>	2.621	2.510	2.450	2.240	2.215	2.188	2.002	2.343	2.05

CJ = Centroid joining,

HT = Hough transform (on centroids),

SVD = Singular value decomposition

4.8.1 Test Environment

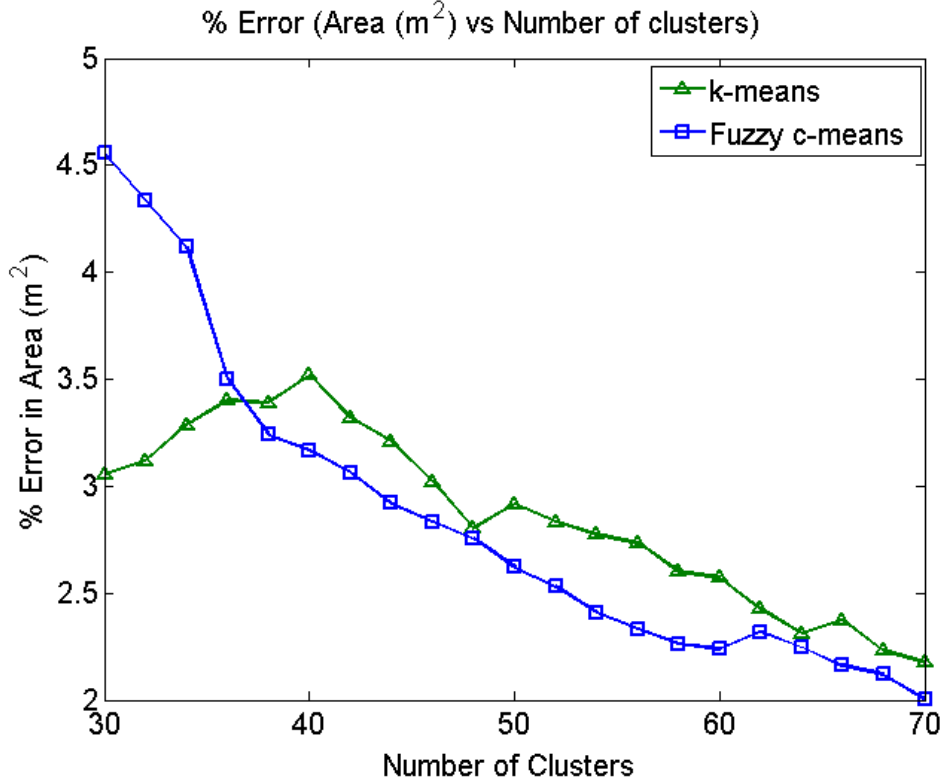


Fig. 4.27 Error plot showing the percentage in error with the number of clusters using k-means and fuzzy c-means clustering.

To test the accuracy and applicability of the proposed algorithms, experiments were performed in both an artificial test environment and a real cluttered environment. The artificial environment is shown in Figure 4.11. The artificial environment is made of straight polygonal walls. The robot was moved in a predefined path and laser data was collected. Random noise was added to test the algorithms. The real environment where the test was conducted consists of long straight corridors with narrow open spaces and cluttered with real objects. Figure 4.31 shows the panoramic view of the real environment. It is cluttered with objects with small narrow spaces for the robot to manoeuvre along. The experiments were performed on an Intel 1.8 GHz Core-i5 system with 4 GB RAM. The C++ language was used for framework implementation and the Java programming language was used for building the GUI.

4.8.2 Mapping Results

Artificial Test Environment

Figure 4.29 shows the final map generated after applying clustering (k-means and c-means) in the spatial domain on noise-free data as shown in Figure 4.13b with a varying number of clusters (30 to 70). Figure 4.27 shows the plot of the percentage error in an area of the map generated using k-means and c-means clustering with an increasing number of centroids. The percentage error is less than five percent with both types of clustering method. It can be seen that c-means clustering gives a slightly better performance compared to k-means clustering. However, the performance advantage diminishes as the number of clusters become larger. Figure 4.30 shows the final map generated from noisy LRF data as shown in Figure 4.13a using clustering in the Hough domain. The calculated percentage error of the map area generated by clustering in the spatial domain by centroid joining, applying a Hough transform on centroids and

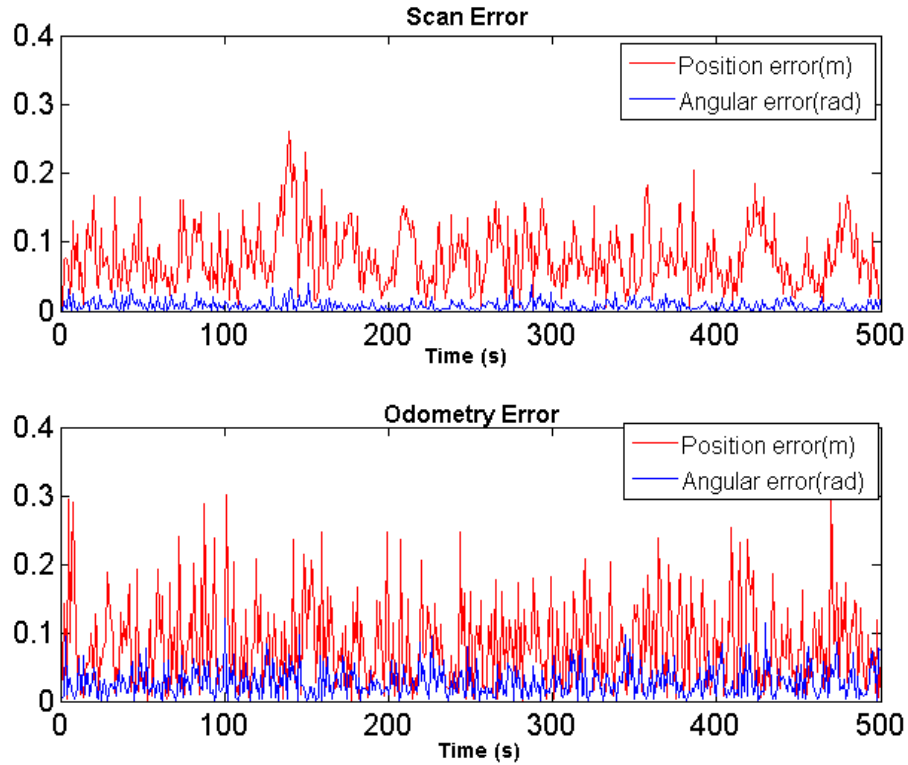
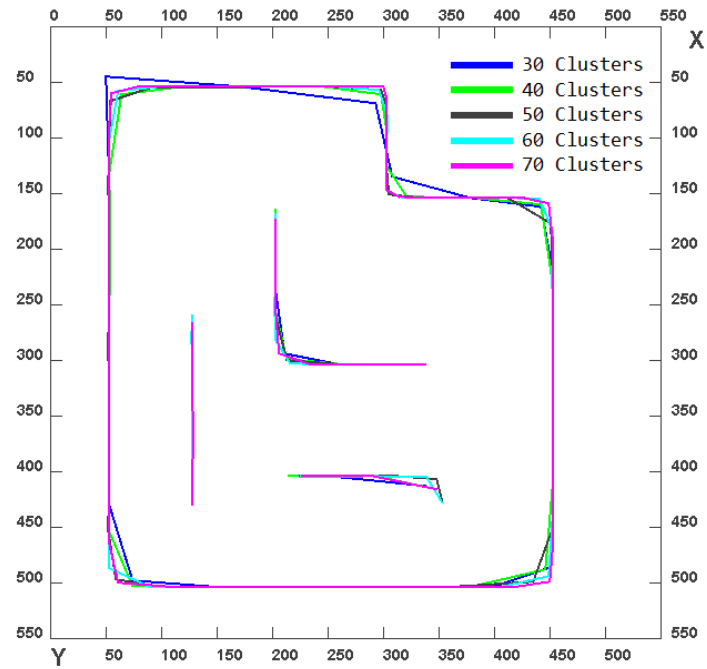
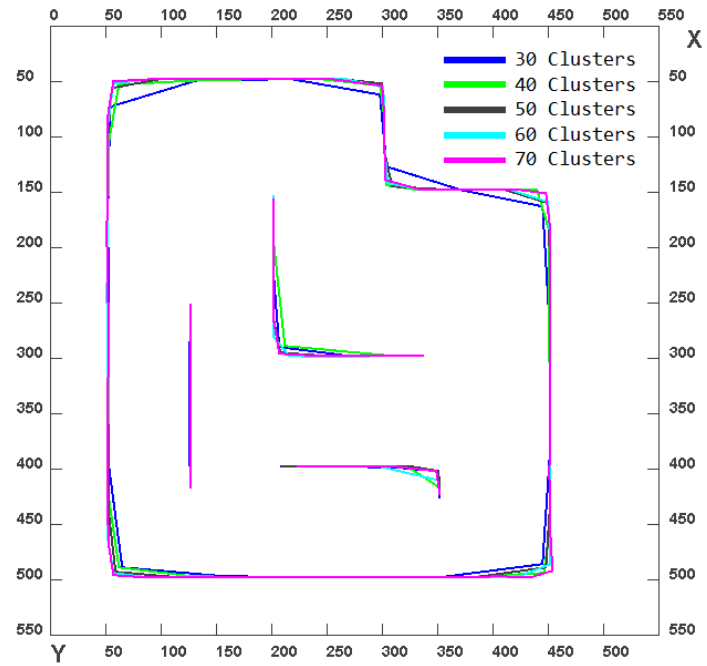


Fig. 4.28 The plots of the scan position error and odometry position error with respect to time.



(a) Map generated using k-means clustering in the spatial domain.



(b) Map generated using c-means clustering in the spatial domain.

Fig. 4.29 Map generated using clustering in the spatial domain.

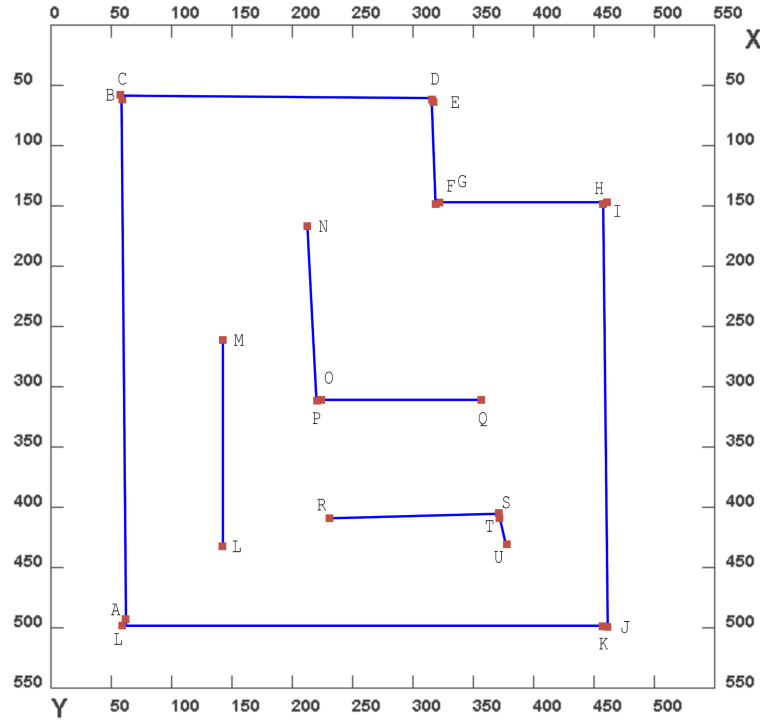


Fig. 4.30 Map generated using clustering in the Hough domain.

by applying SVD on the centroids for 50, 60 and 70 clusters respectively. The results are summarized in Table 3. The fuzzy c-means clustering gives better results than the k-means clustering. Between the Hough transform and SVD on centroids, SVD gives slightly better performance. The plots of the scan position error and the odometry position error with respect to time are shown in Figure 4.28.

Real Environment

To test the robustness of the algorithms, experiments were performed in a real cluttered environment. The standard ICP technique [88] is utilized for fast scan-matching between consecutive scans obtained at different robot poses. In general, ICP is an iterative algorithm that iteratively finds the correspondence between two given scans and calculates a relative transformation that minimizes the distance between the corresponding points until the condition for termination is satisfied. A faster 2D version of the ICP method described in [65] is utilized in this study. For each iteration, it calculates the Euclidean squared distance between each and every possible combination of the registered points in two consecutive scans and finds the correspondence index based on the closest-point rule. The detected outliers are rejected



Fig. 4.31 Actual view of the corridor test environment. The robot was manually tele-operated through narrow passages.

as valid correspondences if their closest distance is above the given threshold. ICP-based localization was utilized in the experiments. The raw laser data for the environment is shown in Figure 4.32. The data has many outliers or noise. Figure 4.33 shows the result of applying DBSCAN on the ICP-corrected laser data. The noise from the raw data is removed after applying DBSCAN. Figure 4.34 shows the result of applying k-means clustering. 160 clusters with their centroids are obtained. The clusters are shown in different colours. Figure 4.35 shows the simple polygon map which is obtained by joining the centroids from the k-means clustering. Figure 4.36 shows a line map generated by applying a Hough transform on the centroids obtained from the k-means clustering algorithm. The result shows straight line maps which are not affected by the cluttered environment. As the Hough transform is applied only on the centroid points, the process is relatively fast. Figure 4.37 gives the result after applying clustering in the Hough domain.

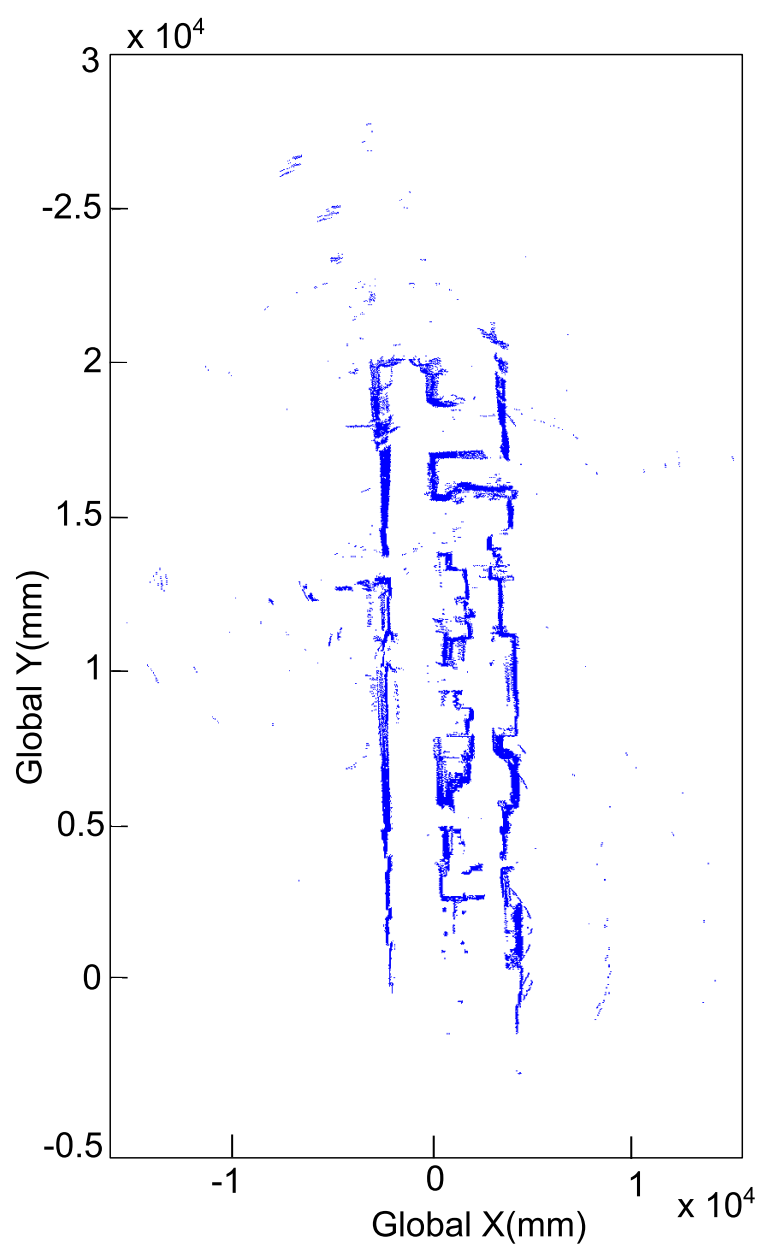


Fig. 4.32 Raw laser scan of the real environment shown in Figure 4.31.

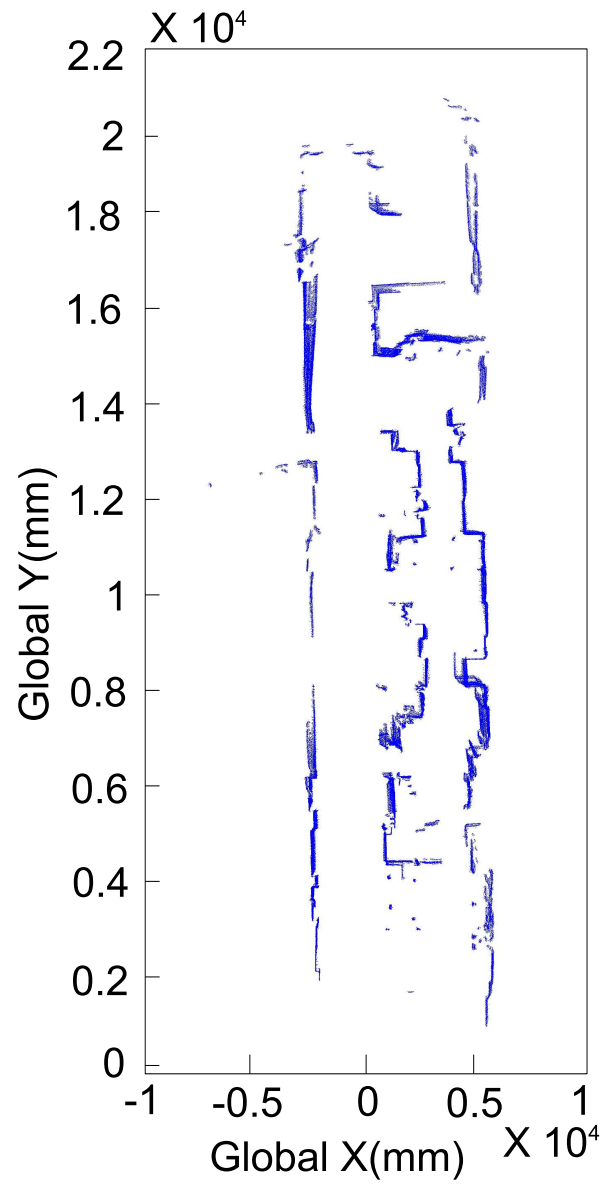


Fig. 4.33 Result of the DBSCAN algorithm on (a).

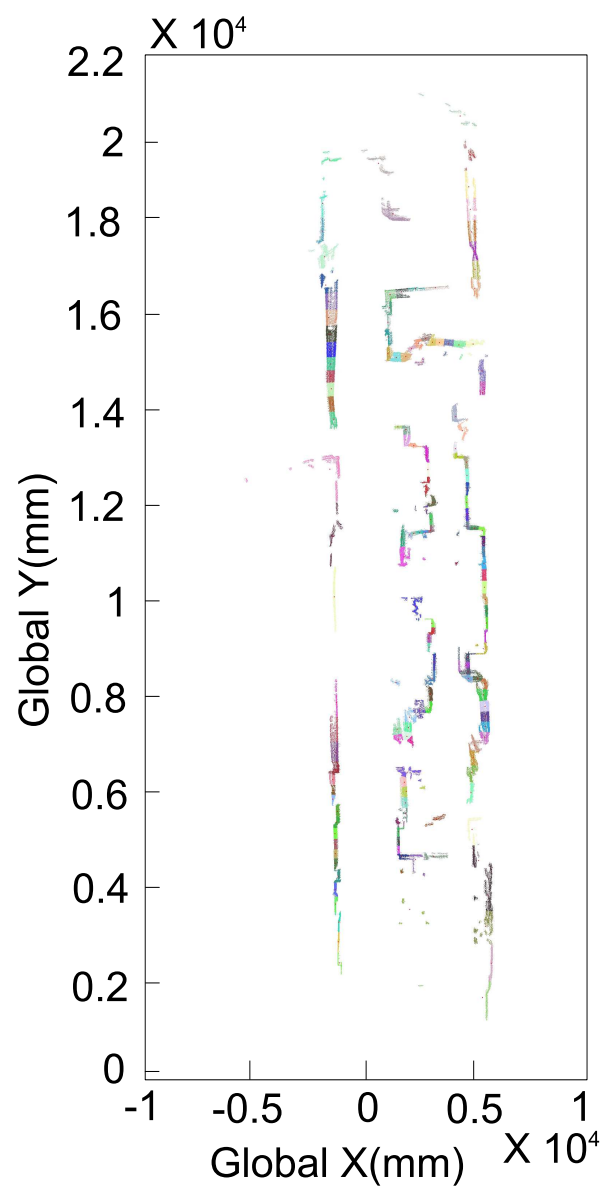


Fig. 4.34 Result of k-means clustering on (b) with clusters and centroids.

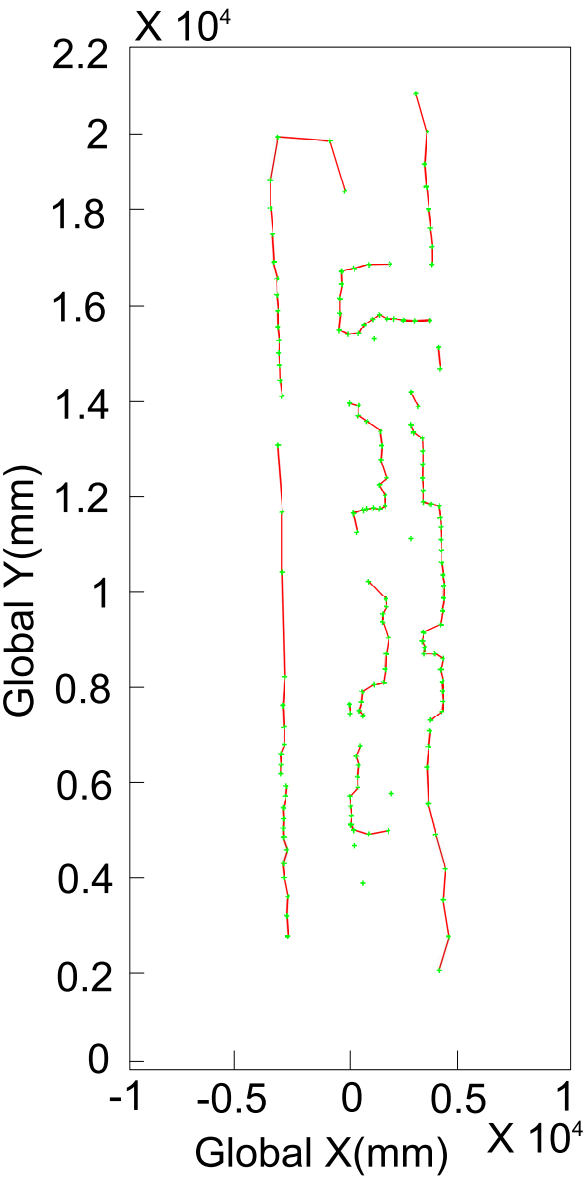


Fig. 4.35 Line maps by joining the centroids of (c).

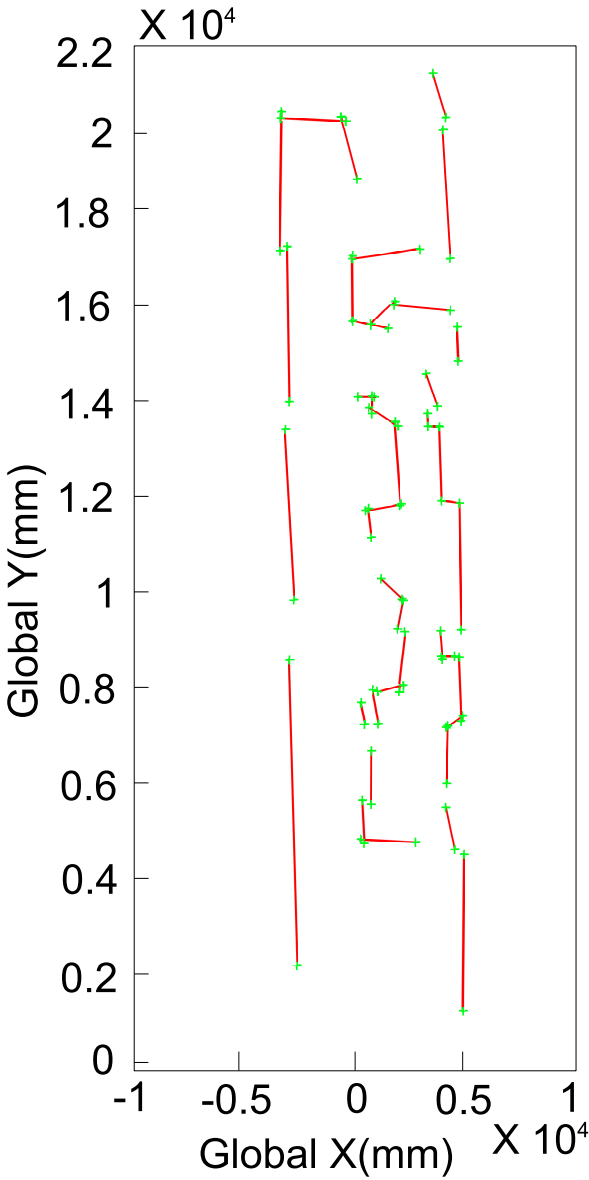


Fig. 4.36 Result of the Hough transform on the centroid points of (c).

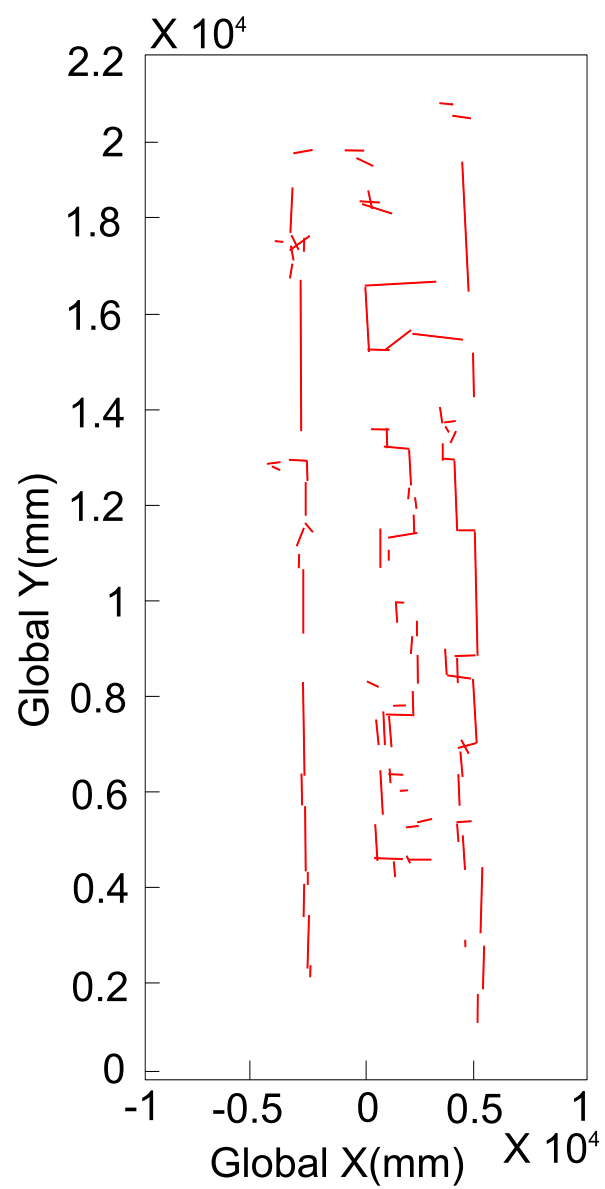


Fig. 4.37 Line map obtained from clustering in the Hough domain. ($\delta = 0$)

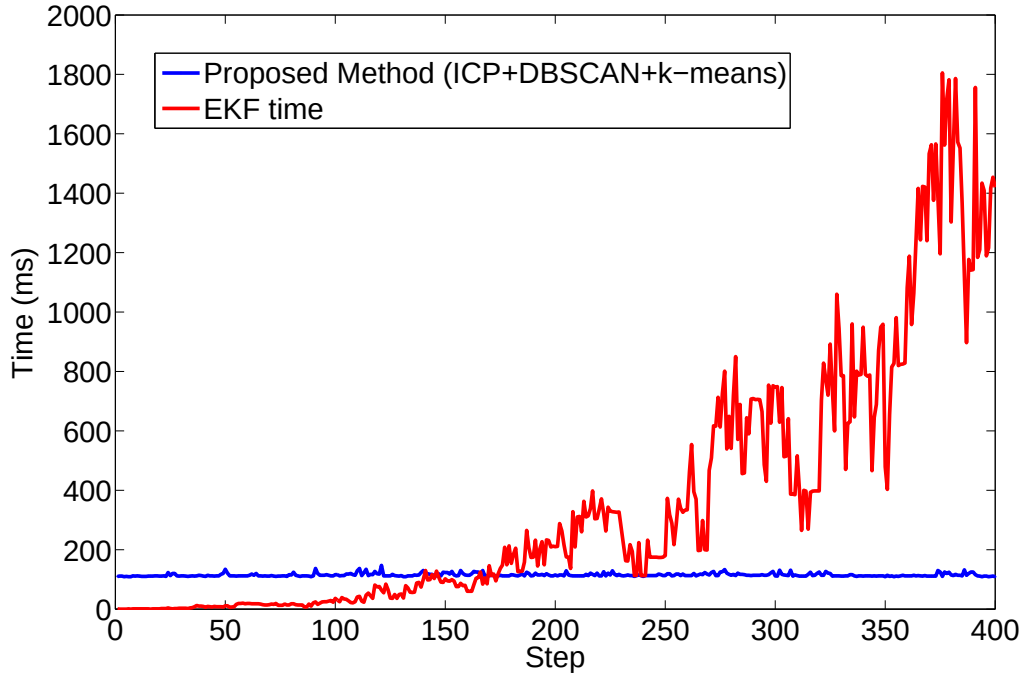


Fig. 4.38 Time cost between EKF-SLAM and the proposed technique.

4.8.3 Comparison with Line Segment-based EKF-SLAM

The proposed algorithms are compared with standard line segment-based EKF-SLAM to check the applicability, improvements in processing time and accuracy of maps generated in an actual cluttered environment. Figure 4.39 shows the line map formed by EKF-SLAM. Due to the nature of the environment where the experiments are performed, noise present in the data affects the line extraction. It can be seen that multiple lines are generated due to poor feature association. Open areas are also not detected correctly and are wrongly associated with line features. EKF-SLAM generally suffers from poor data association because of an increase in uncertainty with increased numbers of features, especially in crowded and cluttered areas. In comparison, the proposed techniques are not affected by any of the above factors and they produce single lines and accurate maps as shown in Figure 4.35, Figure 4.36 and Figure 4.37, which represent the results of a map formed by joining the centroids, performing a Hough transform on centroids and clustering in the Hough domain, respectively.

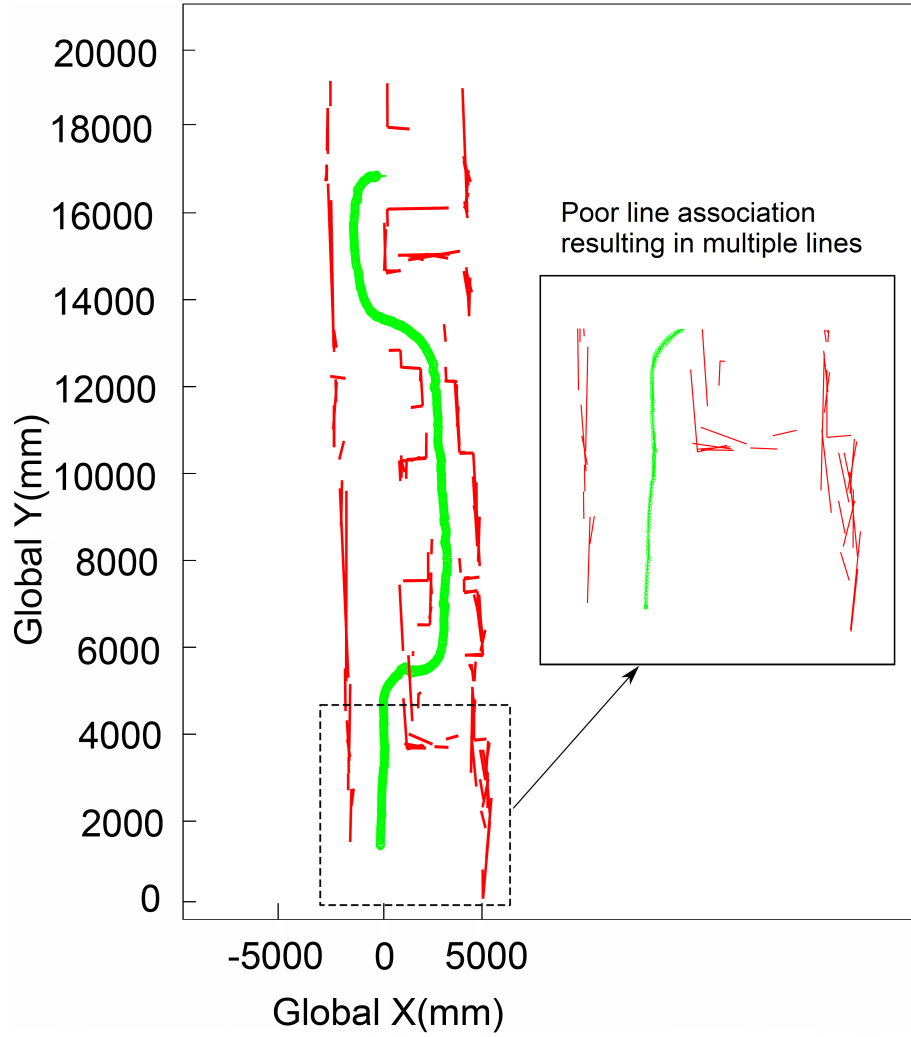


Fig. 4.39 Map generated by line segment-based EKF-SLAM. The EKF robot trajectory is shown in green circles. The inset shows an enlarged part of the figure.

Time Complexity

The time complexity of the proposed technique is compared with EKF-SLAM and the results are shown in Figure 4.38. EKF-SLAM is initially fast when the number of features is lower, but as the features increase, its complexity increases exponentially with time [47]. This is expected as the feature number grows with time, and new features need to be associated by the EKF algorithm. For the complex environment, the uncertainty of the features also grows in the EKF algorithm. On the other hand, with the proposed technique, the map building process involves ICP matching, noise removal, clustering, line extraction and map correction. None of these steps requires complex

feature association and, at each step, only the data that is available is processed. Compared to EKF, the proposed technique is slow, initially, but is more stable and in the long run as well as being more stable than the EKF. This makes the proposed method relatively stable for the entire 400 steps and it is much faster when compared to the EKF.

The time required by the robot to generate two clusters (in one *step*) for k-means & c-means were 26 ms and 38 ms, respectively. For noise removal, Erosion (15 ms) was faster than DBSCAN (28 ms), for one *step* of LRF data. ICP takes about (7 ms), as the robot does not make sudden rotations. Notice that the kernels employed for clustering and the mathematical morphological kernels were not optimized, and their implementation was straightforward. The framework was tested by changing the values of the number of clusters and the type of kernel. The robot stopped for approximately two to three seconds while the new parameters were being loaded.

4.9 Conclusion

This chapter presented algorithms and a framework for generating maps in a noisy environment. Different noise removing techniques are discussed that can effectively remove noise from the LRF data. The maps were generated using clustering techniques in the spatial and Hough domains. In the spatial domain, the algorithm first removes noise from the LRF data, for which it is proposed to use density-based clustering and a mathematical morphological kernel of erosion. A hopping window noise removing technique is proposed that is faster and embarrassingly parallel. Afterwards, clustering was performed and the noise-free data and the centroids obtained were joined to form the map. The results show that straight-line maps with good accuracy can be generated by taking SVD or a Hough transform of the centroids. An algorithm for map building by clustering in the Hough domain was also proposed. Experimental results in both an artificial test environment and a real cluttered environment were performed and the results show that the proposed algorithms are very efficient for robot mapping in a noisy environment. The results of the proposed technique were compared with EKF-SLAM and shows that the proposed techniques can generate accurate single-line maps faster than EKF-SLAM for large datasets. Finally, a framework to load the kernels or change the kernel parameters of the robot is proposed. The proposed framework makes it easy to change the kernels or fine-tune the parameters of the algorithms remotely based on live feedback.

Chapter 5

SLAM- Simultaneous Localization and Mapping

5.1 Introduction

For the mobile robot to map the environment, it is important for it to localize itself in the environment. To do so, there can be two solutions, firstly, there is already a map of the environment and the robot can use the information from the map to estimate its position in that map. Secondly, the robot builds the map of the environment by utilizing the external sensors, and by extracting features from the environment from the LRF data. It can then utilize the map that it has built to localize itself accurately in the environment.

The previous chapter details the feature extraction technique and in this chapter the line features from the LRF data will be used to built upon line segment maps of the environment. It is intended to obtain high degree of accuracy in building the map for accurate localization. And EKF-SLAM solution is utilized for building robust 2D linear feature maps. Without any knowledge of the surrounding the SLAM algorithm allows the robot to estimate the correct position of the robot in the map and the world frame. The EKF-SLAM procedure builds the 2D map of the environment in the first stage. This map is a feature map with line segments and invariant point. In the next stage, EKF with the robot state using the map obtained during the EKF-SLAM stage it is possible to localize the robot. It is assumed that the environment where the robot is operating is static and no dynamic obstacles exist in the map. The robot with the LRF is moved inside the indoor environment and set of LRF data are collected. Next, using the techniques discussed in the previous chapter features are detected and

extracted from the LRF data. Thirdly, EKF-SLAM algorithm is applied using the robot state information and the features extracted.

5.2 Robot Odometry Model

The robot that was used for the experiments is a differential drive robot. The details of the robot can be found in the Appendix section. In general, the pose or the position of the robot is described by the vector

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (5.1)$$

For the differential drive robot the position can be found starting from a known position by integrating the movement. For a discrete system with a fixed sampling interval Δt , the incremental travel distances $(\Delta x, \Delta y, \Delta \theta)$ are

$$\Delta x = \Delta s \cos(\theta + \frac{\Delta \theta}{2}) \quad (5.2)$$

$$\Delta y = \Delta s \sin(\theta + \frac{\Delta \theta}{2}) \quad (5.3)$$

$$\Delta \theta = \frac{\Delta s_r - \Delta s_l}{b} \quad (5.4)$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \quad (5.5)$$

where

$(\Delta x, \Delta y, \Delta \theta)$ = path traveled by in the last sampling interval;

$\Delta s_r, \Delta s_l$ = traveled distances for the right and left wheel respectively;

b = distance between the two wheels. Thus the updated position p' is given by

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos(\theta + \frac{\Delta \theta}{2}) \\ \Delta s \sin(\theta + \frac{\Delta \theta}{2}) \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \frac{\Delta \theta}{2}) \\ \Delta s \sin(\theta + \frac{\Delta \theta}{2}) \\ \Delta \theta \end{bmatrix} \quad (5.6)$$

Using Equation 5.4 and 5.5, we can get

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r - \Delta s_l}{2b} \end{bmatrix} \quad (5.7)$$

The odometric position updates can give only a very rough estimate of the actual position. Owing to integration errors of the uncertainties of p and the motion errors during the incremental motion $(\Delta s_r, \Delta s_l)$ the position error based on odometry integration grows with time.

Next, the error model of the integrated position p' , to obtain the covariance matrix Σ'_p of the odometric position estimate is established. Assuming that the starting point of the initial covariance matrix $\Sigma_{p'}$ is known. For the motion increment we get the covariance matrix Σ_Δ

$$\Sigma_\Delta = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix} \quad (5.8)$$

where, k_r and k_l are the error constants representing the non deterministic parameters of the motor drive and the wheel floor interaction. From Equation (5.8) two assumptions are defined -

- The two errors of the individually driven wheels are independent.
- The variance of the errors (left and right wheels) are proportional to the absolute value of the traveled distances $(\Delta s_r, \Delta s_l)$.

These assumptions, while not perfect, are suitable and will thus be used for the further development of the error model. The motion errors are due to imprecise movement because of deformation of wheel, slippage, unequal floor, errors in encoders, and so on. The values for the error constants k_r and k_l depend on the robot and the environment and should be experimentally established by performing and analyzing representative movements. If we assume p and $\Delta_{rl} = (\Delta s_r, \Delta s_l)$ are uncorrelated and the derivation of $f(x, y, \theta, \Delta s_r, \Delta s_l)$ is reasonably approximated by the first-order Taylor expansion (linearization), we conclude, using the error propagation law

$$\Sigma_{p'} = \nabla_p f \Sigma_p \nabla_p^T f + \nabla_{\Delta_{r,l}} f \Sigma_\Delta \nabla_{\Delta_{r,l}}^T f^T \quad (5.9)$$

The covariance matrix Σ_p is, of course, always given by the $\Sigma_{p'}$ of the previous step, and can thus be calculated after specifying an initial value. Using Equation (5.7), we can get Jacobians, $F_p = \nabla_p f$ and $F_{\Delta_{rl}} = \nabla_{\Delta_{rl}} f$

$$\nabla_p f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta + \Delta\theta/2) \\ 0 & 1 & \Delta s \cos(\theta + \Delta\theta/2) \\ 0 & 0 & 1 \end{bmatrix} \quad (5.10)$$

$$F_{\Delta_{rl}} = \begin{bmatrix} \frac{1}{2} \cos(\theta + \frac{\Delta\theta}{2}) - \frac{\Delta s}{2b} \sin(\theta + \frac{\Delta\theta}{2}) & \frac{1}{2} \cos(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2b} \sin(\theta + \frac{\Delta\theta}{2}) \\ \frac{1}{2} \sin(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2b} \cos(\theta + \frac{\Delta\theta}{2}) & \frac{1}{2} \sin(\theta + \frac{\Delta\theta}{2}) - \frac{\Delta s}{2b} \cos(\theta + \frac{\Delta\theta}{2}) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix} \quad (5.11)$$

Since,

$$\begin{aligned} F_{\Delta_{rl}} &= \nabla_{\Delta_{rl}} f \\ &= \begin{bmatrix} \frac{\partial f}{\partial \Delta s_r} & \frac{\partial f}{\partial \Delta s_l} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \Delta s}{\partial \Delta s_r} \cos(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2} - \sin(\theta + \frac{\Delta\theta}{2}) \frac{\partial \Delta\theta}{\partial \Delta s_r} & \frac{\partial \Delta s}{\partial \Delta s_l} \cos(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2} - \sin(\theta + \frac{\Delta\theta}{2}) \frac{\partial \Delta\theta}{\partial \Delta s_l} \\ \frac{\partial \Delta s}{\partial \Delta s_r} \sin(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2} \cos(\theta + \frac{\Delta\theta}{2}) \frac{\partial \Delta\theta}{\partial \Delta s_r} & \frac{\partial \Delta s}{\partial \Delta s_l} \sin(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2} \cos(\theta + \frac{\Delta\theta}{2}) \frac{\partial \Delta\theta}{\partial \Delta s_l} \\ \frac{\partial \Delta\theta}{\partial \Delta s_r} & \frac{\partial \Delta\theta}{\partial \Delta s_l} \end{bmatrix} \end{aligned} \quad (5.12)$$

with,

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} ; \quad \Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \quad (5.13)$$

and,

$$\frac{\partial \Delta s}{\partial \Delta s_r} = \frac{1}{2} ; \quad \frac{\partial \Delta s}{\partial \Delta s_l} = \frac{1}{2} ; \quad \frac{\partial \Delta\theta}{\partial \Delta s_r} = \frac{1}{b} ; \quad \frac{\partial \Delta\theta}{\partial \Delta s_l} = -\frac{1}{b} \quad (5.14)$$

Once the error model has been established, the error parameters must be specified. One can compensate for deterministic errors properly calibrating the robot. However the error parameters specifying the nondeterministic errors can only be quantified by statistical (repetitive) measurements. A detailed discussion of odometric errors and a method for calibration and quantification of deterministic and nondeterministic errors can be found in [12].

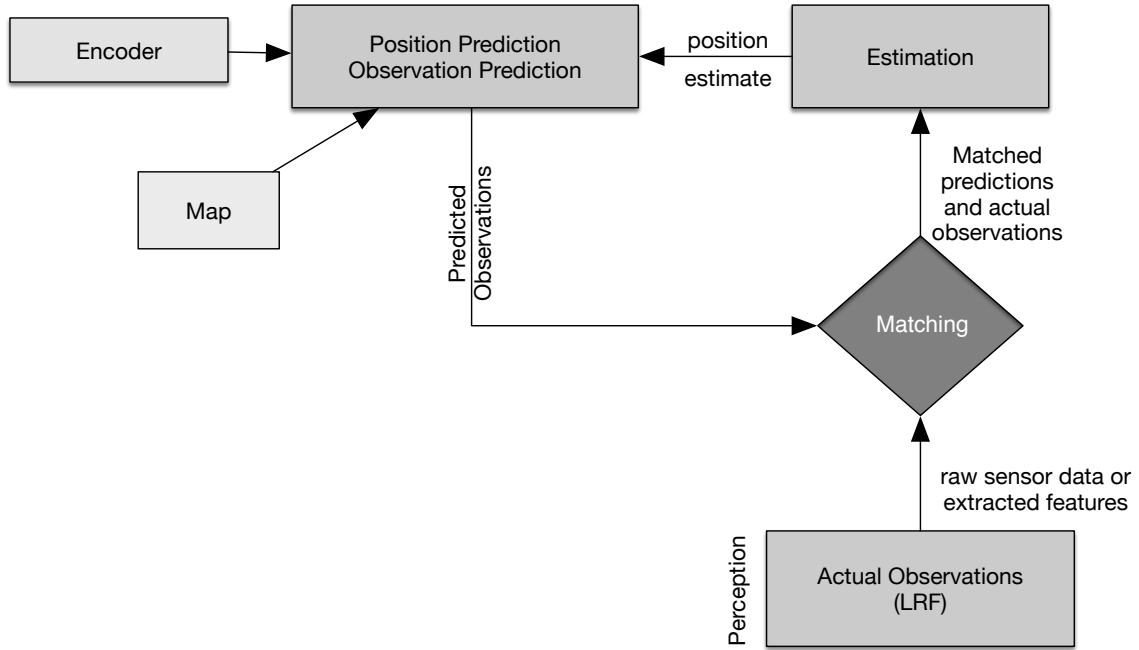


Fig. 5.1 Schematic flow of the Kalman filter localization [58]

5.3 Kalman Filter localization - line feature extraction

The last chapter briefly described the line extraction and detection techniques from the LRF data. Now, an Extended Kalman Filter based Line SLAM will be implemented utilizing those linear features along with the robot state updates. The process for the EKF localization takes place in five steps[91]:

1. **Robot Position Prediction.** At time k , robot is at position $p(k) = [x(k) \ y(k) \ \theta(k)]^T$ and its best position estimates is $\hat{p}(k|k)$. The control input $u(k)$ drives the robot to the position $p(k+1)$. The robot position prediction $\hat{p}(k+1)$ and the time increment $k+1$ can be computer from the previous estimate

$\hat{p}(k|k)$ and the odometric integration of the movement. For the differential drive robot the robot odometry model has been discussed in Section 5.2.

$$\begin{aligned}\hat{p}(k+1|k) &= \hat{p}(k|k) + u(k) \\ &= \hat{p}(k|k) + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}\end{aligned}\quad (5.15)$$

with updated covariance matrix

$$\Sigma_p(k+1|k) = \nabla_p f \cdot \Sigma_p(k|k) \cdot \nabla_p f^T + \nabla_u f \cdot \Sigma_u(k) \cdot \nabla_u f^T \quad (5.16)$$

where

$$\Sigma_u = \text{cov}(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix} \quad (5.17)$$

where

$$\Sigma_u = \text{cov}(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix} \quad (5.18)$$

2. **Observation.** For line based localization, each single observation (line feature) is extracted from the raw LRF data and consists of two line parameters $\beta_{0,j}$, $\beta_{1,j}$ or α_j , r_j respectively. For the rotating LRF, the polar cartesian format is used and given by

$$z_j(k+1) = \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix}_R \quad (5.19)$$

After getting the raw data at time $k+1$, lines and their uncertainties are extracted. This leads to n_0 observed lines with $2n_0$ line parameters and a covariance matrix for each line that can be calculated from the uncertainties of all the measurement points contributing to each line as developed for line extraction in the chapter 3.

$$\Sigma_{R,j} = \begin{bmatrix} \sigma_{\alpha\alpha} & \sigma_{\alpha r} \\ \sigma_{r\alpha} & \sigma_{rr} \end{bmatrix}_j \quad (5.20)$$

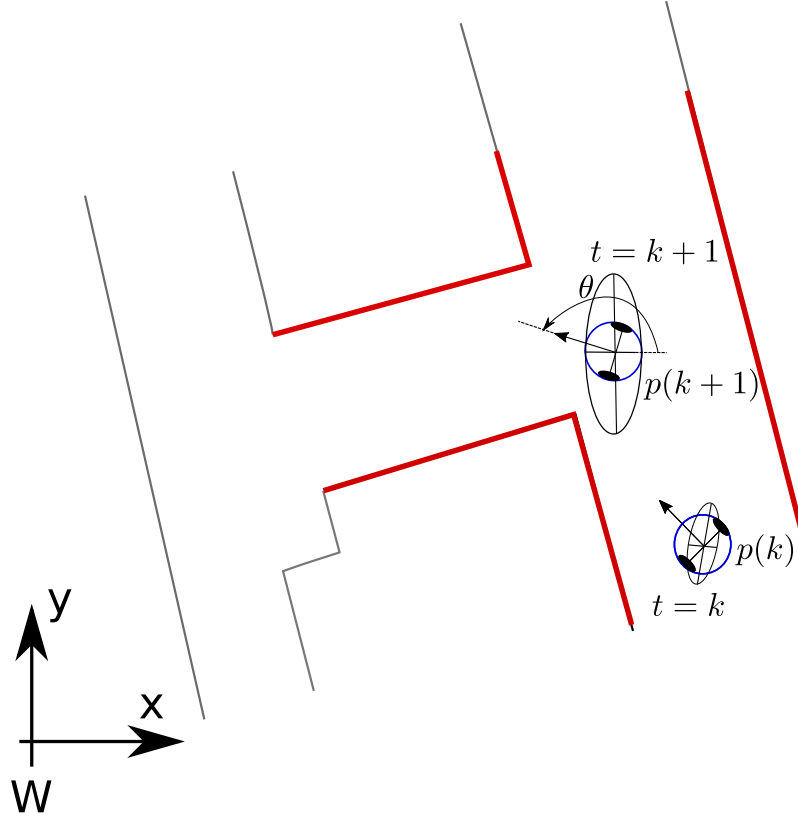
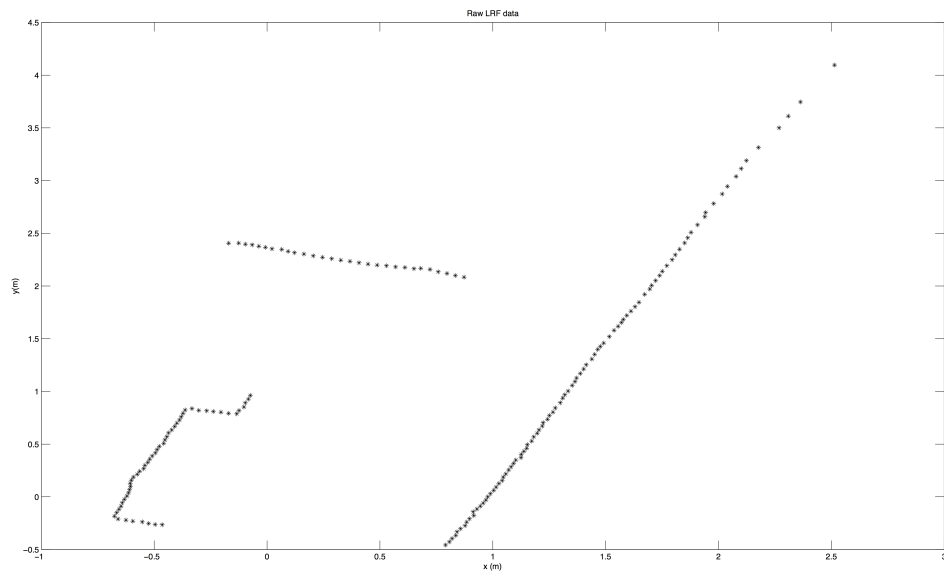
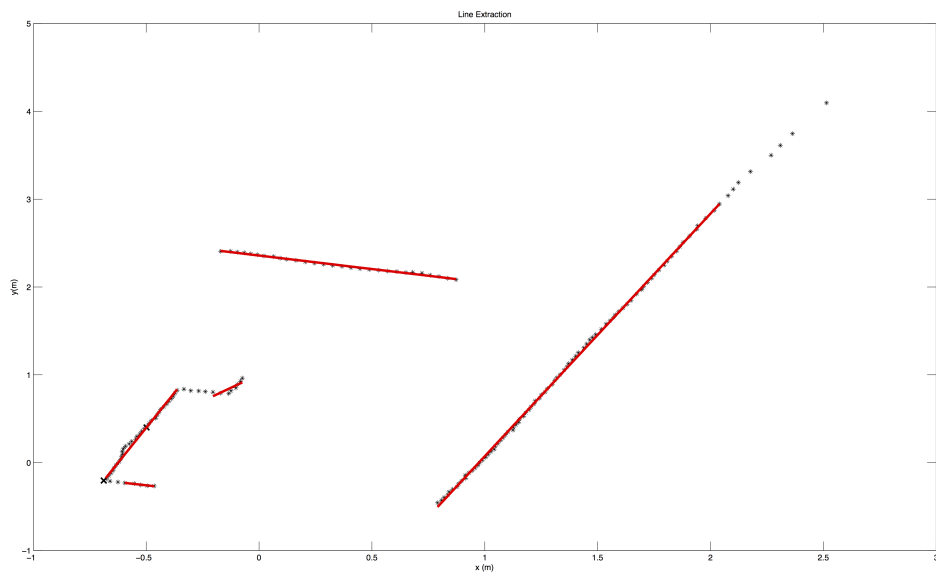


Fig. 5.2 Prediction of the robot position(red) based on former position. Ellipses represents the uncertainty in x, y direction.

3. **Measurement prediction.** Based on the stored map and the predicted robot position $\hat{p}(k|k)$, the measurement predictions of expected features $z_{t,i}$ are generated. To reduce the required calculation power, there is often an additional step that first selects the possible features (line segments), from the whole set of features in the map. These lines are stored in the map and specified in the world coordinate system (W). Therefore they need to be transformed to the



(a) A raw laser scan of the actual environment



(b) Line extracted from the laser data.

Fig. 5.3 Observation: From the raw LRF data

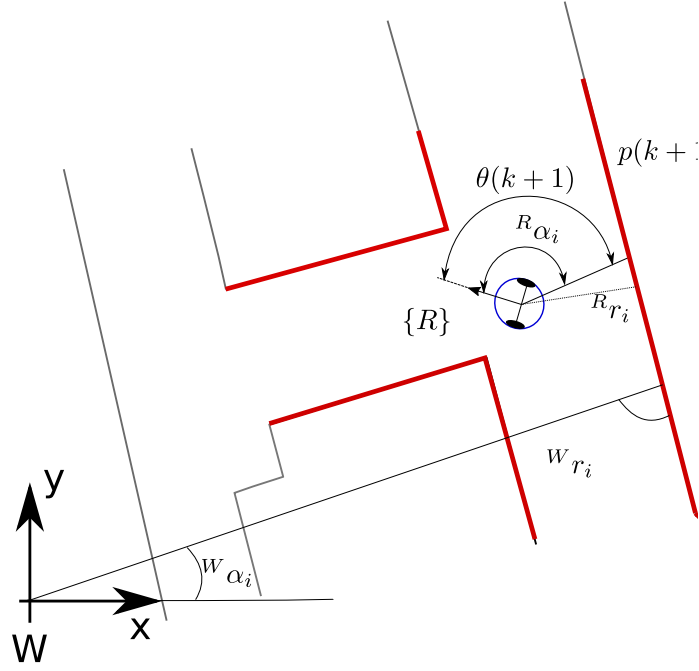


Fig. 5.4 Target position representation in world and robot frame

robot frame (R).

$${}^W z_{t,i} = \begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix}_W \rightarrow {}^R z_{t,i} = \begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix}_R \quad (5.21)$$

The transformation is given by ,

$$\begin{aligned} \hat{z}_t(k+1) &= \begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix}_R \\ &= h_t(z_{t,t}, \hat{p}(k+1|k)) \\ &= \begin{bmatrix} {}^W \alpha_{t,i} - {}^W \hat{\theta}(k+1|k) \\ {}^W r_{t,i} - ({}^W \hat{x}(k+1|k) \cos({}^W \alpha_{t,i}) + {}^W \hat{y}(k+1|k) \sin({}^W \alpha_{t,i})) \end{bmatrix} \end{aligned} \quad (5.22)$$

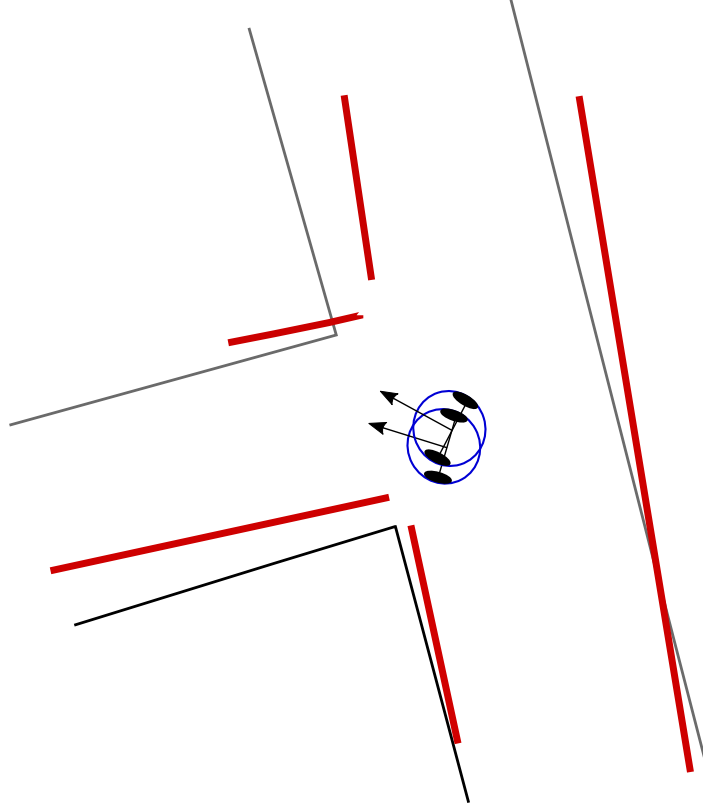


Fig. 5.5 Kalman filter matching. Observations (red) and measurement prediction (black) are matched and the innovation and its uncertainties are calculated

and the Jacobian ∇h_i is given by,

$$\begin{aligned} \nabla h_i &= \begin{bmatrix} \frac{\partial \alpha_{t,i}}{\partial \hat{x}} & \frac{\partial \alpha_{t,i}}{\partial \hat{y}} & \frac{\partial \alpha_{t,i}}{\partial \hat{\theta}} \\ \frac{\partial r_{t,i}}{\partial \hat{x}} & \frac{\partial r_{t,i}}{\partial \hat{y}} & \frac{\partial r_{t,i}}{\partial \hat{\theta}} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & -1 \\ -\cos^W \alpha_{t,i} & -\sin^W \alpha_{t,i} & 0 \end{bmatrix} \end{aligned} \quad (5.23)$$

The measurement prediction results in predicted lines represented in the robot coordinate frame. These measurements are uncertain since the robot position is also uncertain.

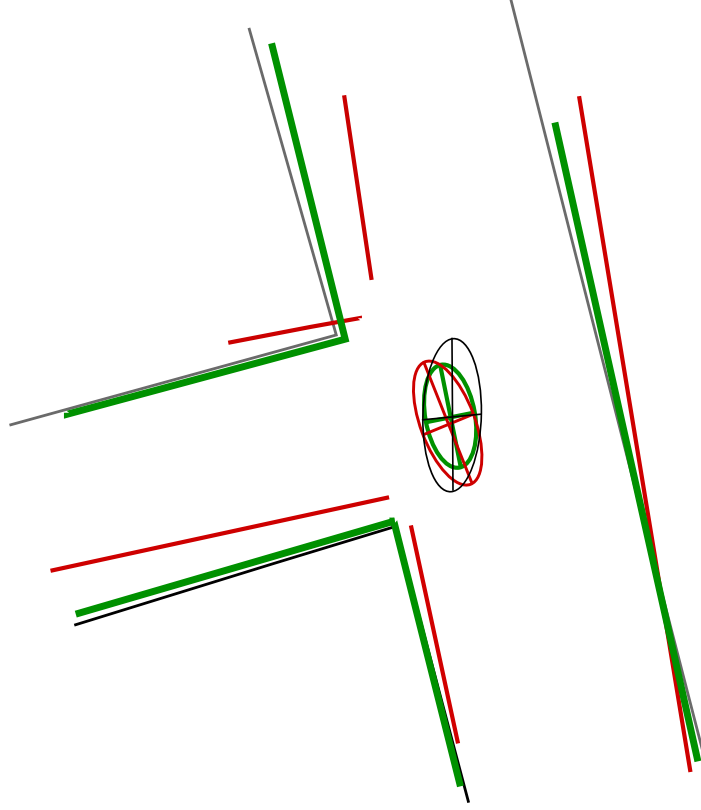


Fig. 5.6 Kalman filter estimation of the position. Fusing the prediction of robot position (black) with the innovation gained by the measurements (red) we get the updated estimate of the robot position (green)

4. **Matching.** In the matching step, the correspondences between the predicted and observed features are found out. To do that, Mahalanobis distance are used

$$\nu_{ij}^T(k+1) \cdot \Sigma_{IN,ij}^{-1}(k+1) \cdot \nu_{ij}(k+1) \leq g^2 \quad (5.24)$$

with

$$\begin{aligned} \nu_{ij}(k+1) &= [z_j(k+1) - h_i(z_t, \hat{p}(k+1|k))] \\ &= \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix} - \begin{bmatrix} {}^W\alpha_{t,i} - {}^W\hat{\theta}(k+1|k) \\ {}^Wr_{t,i} - ({}^W\hat{x}(k+1|k) \cos({}^W\alpha_{t,i}) + {}^W\hat{y}(k+1|k) \sin({}^W\alpha_{t,i})) \end{bmatrix} \end{aligned} \quad (5.25)$$

$$\Sigma_{IN,ij}(k+1) = \nabla h_i \cdot \Sigma_p(k+1|k) \cdot \nabla h_i^T + \Sigma_{R,t}(k+1) \quad (5.26)$$

to enable finding the best matches while eliminating all other remaining observed and predicted unmatched features.

5. **Estimation.** Applying the Kalman filter results in a final pose estimate corresponding to the weighted sum of

- the pose estimates of each matched pairing of observed and predicted features.
- the robot position estimation based on odometry and observation positions.

These corrected features are continuously added to new feature matrix and the unmatched features are removed. The reason for the SLAM problem to be difficult is because of the interaction between the robot's position updates as it is localizing and mapping. If the robot localizes its position based on any imperfect feature the resultant position estimate will become correlated with the feature location estimate. Similarly, the map becomes correlated with the position estimate if an observation taken from an imprecise position is used to update or add a feature to the map. For localization the robot needs to know where the features are, and for map building the robot needs to know where it is on the map. In order to do that, the robot has to incrementally build the map and localize itself in the map. Figure 5.7 shows the process flow for the localization and map building process for SLAM. The additional flow of information when there are imperfect match between the observations and measurement predictions are shown in the flowchart .

Unexpected observations will effect the creation of new features in the map, whereas unobserved measurement predictions will effect the removal of features from the map. As discussed earlier, each specific prediction or observation has an unknown exact value and so it is represented by a distribution. The uncertainties of all of these quantities must be considered throughout this process.

The new map is represented by $\mathcal{M}$ with a set of n of probabilistic feature locations $\hat{z}_t$, each with covariance matrix Σ_t and an associated credibility factor c_t between 0 and 1 which quantifies the belief in the existence of the feature in the environment.

$$\mathcal{M} = \{\hat{z}_t, \Sigma_t, c_t | (1 \leq t \leq n)\} \quad (5.27)$$

Here the map $\mathcal{M}$ is not assumed to be precise as it is being build by an uncertain robot over time. Hence, the features $\hat{z}_t$ are described with associated covariance

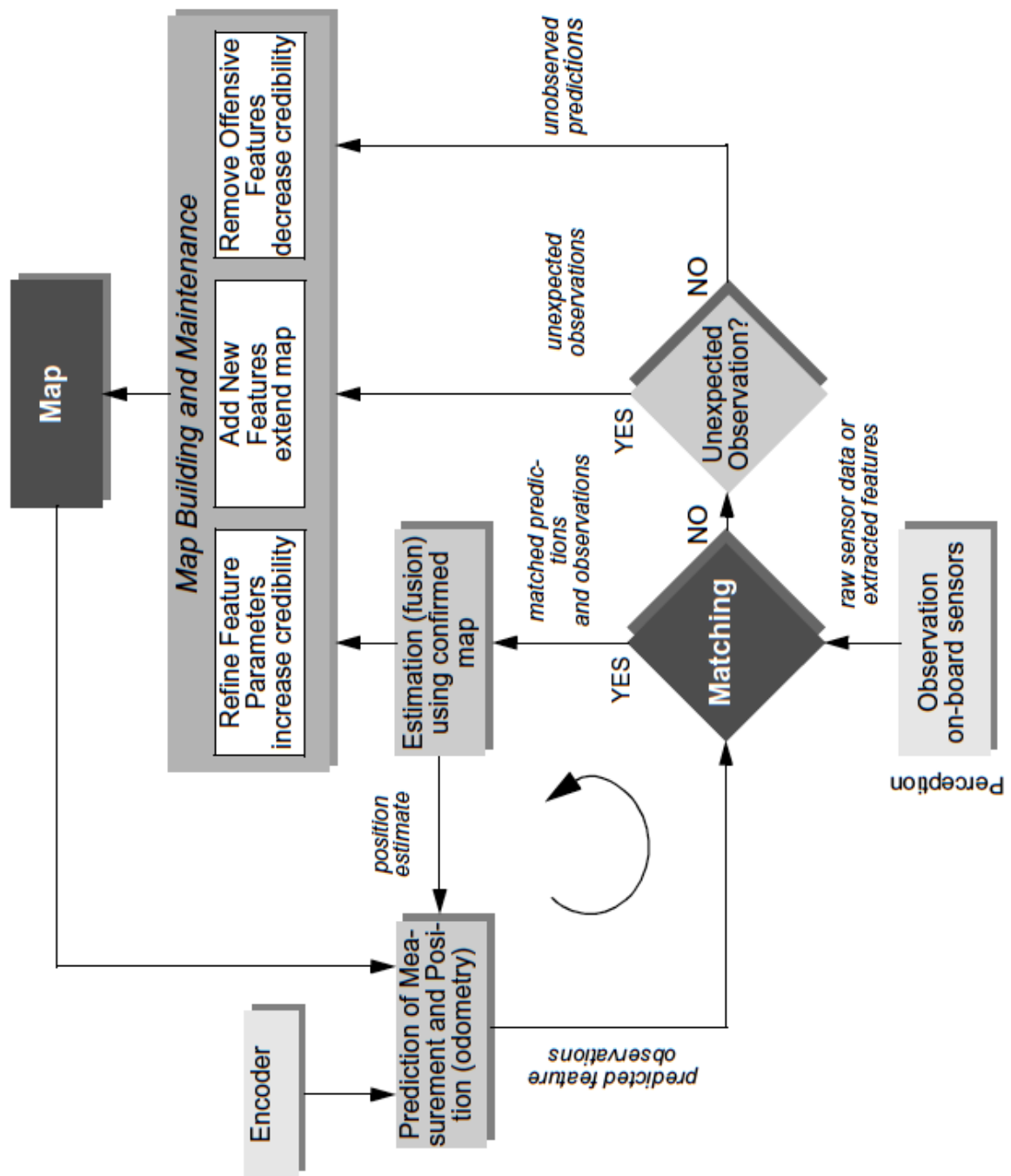


Fig. 5.7 Schematic flow of the SLAM process [58, 91]

matrices Σ_t . The credibility factor is given by [58].

$$c_t(k) = 1 - e^{-\left(\frac{n_s}{a} - \frac{n_u}{b}\right)} \quad (5.28)$$

where a and b define the learning and forgetting rate and n_s and n_u are the number of matched and unobserved predictions up to time k , respectively. The update of the covariance matrix Σ_t can be done similarly to the position update seen previously. In map building the feature position and robot positions are strongly correlated, it is thus, important to update the map with all cross-correlations in a map given by

$$X = \begin{bmatrix} x_r(k) & x_1(k) & \cdots & x_n(k) \end{bmatrix} \quad (5.29)$$

and the system covariance matrix is given by

$$\Sigma = \begin{bmatrix} C_{rr} & C_{r1} & C_{r2} & \cdots & C_{rn} \\ C_{1r} & C_{11} & C_{12} & \cdots & C_{1n} \\ C_{2r} & \cdots & \cdots & \cdots & C_{2n} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ C_{nr} & C_{n1} & C_{n2} & \cdots & C_{nn} \end{bmatrix} \quad (5.30)$$

where r stands for the robot and $i = 1$ to n for the features in the map. This matrix will continue to grow as more and more features are extracted and updated to build the map and thus shows why the SLAM problem is difficult to handle when the feature number increases.

So far, the mapping features that were considered in this thesis only consisted on simple feature parameters (corners, line segment). This works well in a structured, non cluttered environment with orthogonal features and depending on the precision of odometry data that the SLAM algorithm receives quality maps can be generated from these features. This however is not the case in actual implementation. Most of the indoor environments are cluttered with objects. The robot model and the sensor model are not accurate and generates drift in the data over time. Many of the recent mapping techniques in the SLAM community therefore use what is called as the Occupancy Grid Maps (OGM). The next section will describe the theory and implementation of the OGM on the LRF sensor data with extensions to grid based mapping using particle filters and Rao-Blackwellized particle filters for more efficient maps.

5.4 Occupancy Grid Mapping (OGM)

Occupancy grid maps addresses the problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known. The basic idea of the occupancy grid is to represent the map as a field of random variables, arranged in an evenly spaced grid. Each random variable is binary and corresponds to the occupancy of the location it covers. OGM algorithm implement approximate posterior estimation for those random variables. Since the complete map can be described over a grid with occupied cells (0 being unoccupied and 1 being occupied), the memory to store the map is significantly low compared to other techniques of storing maps (for eg. point maps or line maps).

Occupancy Grid Mapping Algorithm

The OGM algorithm calculates the posterior over maps given the data

$$p(m|z_{1:k}, x_{1:k}) \quad (5.31)$$

here m is the map, $z_{1:k}$ are the set of measurements over time k and $x_{1:k}$ are the poses of the robot.

Let m_i denote the grid cell with index i . An OGM partitions the space into finely many grids

$$m = \sum_i m_i \quad (5.32)$$

Each m_i has attached to it a binary occupancy value, which specifies whether a cell is occupied or free(0 or 1). The probability that the cell is occupied is given by $p(m_i)$.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems namely estimating

$$p(m_i|z_{1:k}, x_{1:k}) \forall m_i \quad (5.33)$$

This enables us to represent dependencies among neighboring cells, instead the posterior over maps is approximated as the product of its marginals

$$p(m|z_{1:k}, x_{1:k}) = p(m_i|z_{1:k}, x_{1:k}) \quad (5.34)$$

For this factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with static state. The OGM algorithm uses the

log-odds representation of occupancy given by

$$l_{k,i} = \log \frac{p(m_i|z_{1:k}, x_{1:k})}{1 - p(m_i|z_{1:k}, x_{1:k})} \quad (5.35)$$

The probabilities are easily recovered from the log-odds ratio

$$p(m_i|z_{1:k}, x_{1:k}) = 1 - \frac{1}{1 + \exp\{l_{k,i}\}} \quad (5.36)$$

Algorithm 3: Occupancy Grid Mapping algorithm

Data: Occupancy Grid Mapping algorithm($\{l_{k-1,i}, x_k, z_k\}$)

```

1 for all cells  $m_i$  do
2   if  $m_i$  in perceptual field of  $z_k$  then
3      $l_{k,i} = l_{k-1,i} + \text{InverseSensorModel}(m_i, x_k, z_k) - l_0$ 
4   else
5      $l_{k,i} = l_{k-1,i}$ 
6   end
7 end
8 return  $\{l_{k,i}\}$ 

```

The OGM algorithm is given in Algorithm 3. It loops through all the grid cells i , and updates those that fall into the sensor cone of the measurement z_k . The occupied cells are updated by the virtue of inverse sensor model [99]. The constant l_0 is the prior of occupancy represented as

$$l_0 = \log \frac{p(m_i = 1)}{p(m_i = 0)} = \log \frac{p(m_i)}{1 - p(m_i)} \quad (5.37)$$

The inverse sensor model implements the inverse measurement model $p(m_i|z_k, x_k)$ in log-odd form

$$\text{InverseSensorModel}(m_i, x_k, z_k) = p(m_i|z_k, x_k) \quad (5.38)$$

Figure 5.8 shows an example of the OGM constructed from the LRF data. Also shown is the raw odometry scan of the environment. Because of the limitations of the EKF SLAM due to restrictions to landmark based maps which might not be suitable for long term mapping or when mapping areas that contain several thousand's of features. The OGM on the other hand, is much simpler to maintain and is a favorite choice

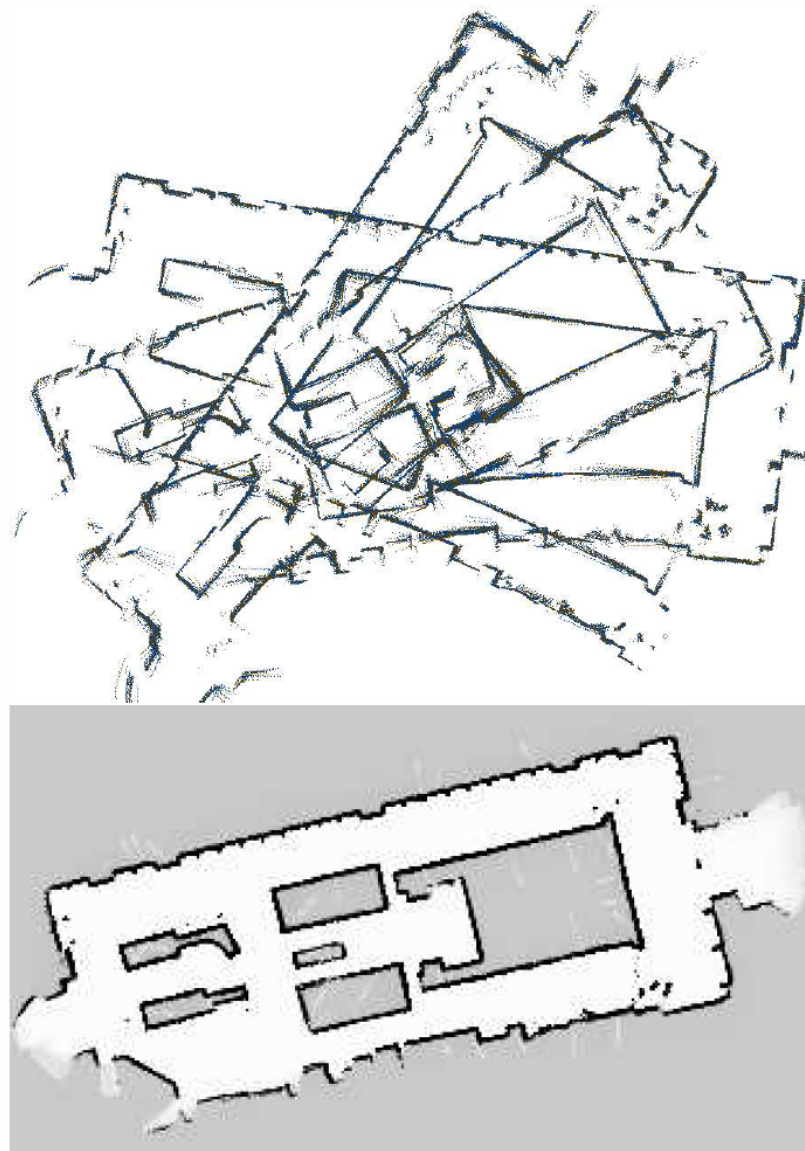


Fig. 5.8 Raw odometry data and Occupancy Grid map[99]

among SLAM researchers for its memory efficiency and its ability to handle large data that can be parsed into simple occupied cells. Another advantage of using OGM over feature maps is that it can be modeled for different types of sensors including low cost ultrasound sensors and infrared range sensors [26, 98]. Recently, 3D range sensors are also being modeled using the OGM in various applications ranging from mobile robot navigation and aerial robots[67, 72].

5.5 Graph SLAM

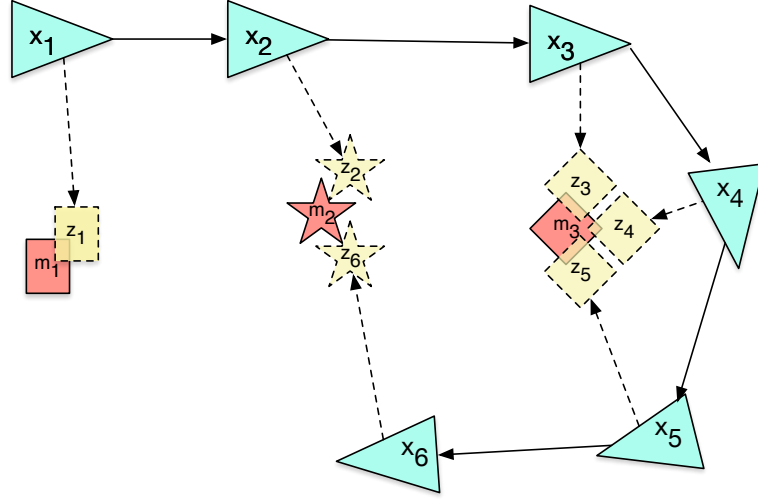


Fig. 5.9 Schematic representation of a graph and the resulting constraints. Solid arrows occur between successive poses and indicate a weak link. Equally shaped measurements z_k are referred to an observation of the same landmark. Those common observation indicate a strong link between the corresponding poses.

A popular technique for the solution of the full SLAM problem is referred to a Maximum-Likelihood approach, in which the robot's trajectory is optimized iteratively. It is based on a graph, in which each node is labeled with a robot pose and the corresponding observation[39, 99], hence the name graph-based SLAM. An exemplaric situation is illustrated in Figure 5.9 in which equally shaped figures, labeled with z_k , represent observations of the very same feature. The criterion to measure the quality of a pose configuration can be derived by a number of spatial constraints between poses, which are represented as edges inside the graph. One usually subdivides those links between poses into weak and strong links [61].

- **Weak Links** occur between successive poses and represent information, that was extracted from odometry readings or control variables. In Figure 5.9 a weak link is available, wherever a solid arrow connects two poses.
- **Strong Links** arise, where parts of the environment have been observed from at least two different poses. In this particular example, poses from whom the same landmark was observed, are linked in this way.

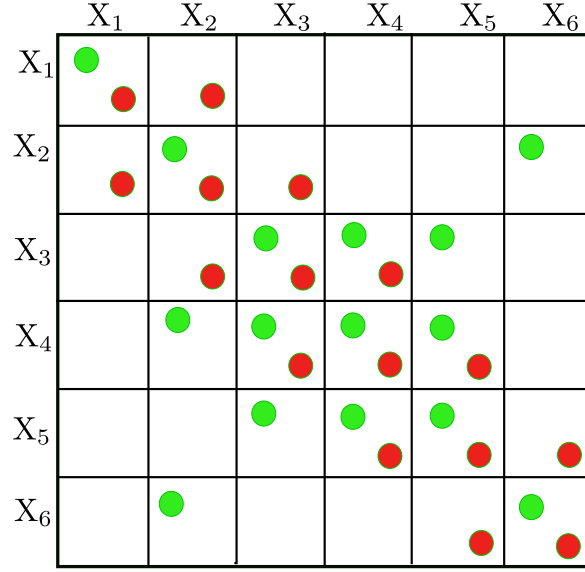


Fig. 5.10 Strong and weak links, as they occur in the example above. Green marks refer to a strong link, red ones to a weak link. Weak links occur only between succeeding poses, strong links, whenever a common part of the environment has been observed from two different poses. Each pose is linked to itself via a strong and a weak link.

The existence of weak and strong constraints in the particular example from Figure 5.9 is shown in Figure 5.10.

In general, graph-based SLAM is referred to a recursive two-step procedure. Defining and integrating the constraints is subject to the so-called *front-end* [39, 54] and leads to an objective function of the following form:

$$F(x_1, x_2, \dots, x_k) = \sum_{(i,j)} e_{i,j}^T \Omega_{i,j} e_{i,j} \quad (5.39)$$

where, $\Omega_{i,j}$ is a known information matrix and $e_{i,j}$ some difference between estimated and observed relative pose values. Optimization of Equation 5.39 is addressed by the *back-end* step [39, 54]. A number of algorithms exists for both steps [39, 54, 61, 99].

Due to the fact, that graph-based SLAM re-estimates the map in each step, it overcomes the restriction of updating individual parts of the map independently, what is particularly beneficial in the context of loop-closure [61, 99]. A major disadvantage lies in its high memory allocation, that occurs because observations have to be stored and can not be rejected, as it is the case e.g. in EKF-SLAM [99]. Furthermore, a bad initial trajectory estimate may cause the algorithm to converge slow and to stuck in local minima [90]. For purposes of easier visualization and explanation, the map

representation has been chosen to be landmark-based here. Basically, graphSLAM is able to deal with different kinds of maps as well. Examples on how to use it on grid-based maps can be found, among others in [54] and [61].

5.6 Experiment Results

Experiments were conducted in real environment narrow corridors and cluttered with objects in the graduate school of Engineering. Two scenarios were chosen to test the mapping algorithms.

1. *Scenario 1:* The first experiment area consists of long straight walls in a long corridor with objects that add to the generation of noise from LRF data. The robot is equipped with a horizontal laser sensor and was manually tele-operated to collect the LRF scans and odometry data.

The total length of distance traveled by the robot is about 260 m. In the first step, the robot is slowly maneuvered across the corridor and the LRF scans are stored. The noise generated from the data was removed from the LRF data using clustering techniques as described in Chapter 4. For this experiment DBSCAN was chosen as the noise removing algorithm due to its speed and ability to cluster noise areas based on density. For extracting the lines, the chosen method was clustering of Hough domain space parameters. For filtering the standard EKF method was used. Figure 5.11 shows the result of the EKF based line segment map of the environment. Total 203 line features were extracted from the EKF mapping with 278 invariant points. A map with this dimension introduces a lot of features in the Extended Kalman Filter, which makes the EKF-SLAM impracticable to operate online. The red dotted line is the raw odometry generated by the robot and the green line is the EKF corrected trajectory. The difference between the trajectories happens due the continuous map's correction, in the EKF-SLAM cycle, which happens in the map vector, when there are a closure loop. When applied the EKF-Localization, the map remains static, therefore it produces a different trajectory. The pre-localization is performed only once and offline, as preparation stage to obtain the distance and gradients matrices used during the localization. Although, the absence of restrictions in terms of spent time during the pre-localization and mapping procedures, when the entire map is divided, into smaller maps, the number of mapped features decreases, reducing the computational time spent. Besides that, when the EKF-SLAM procedure is

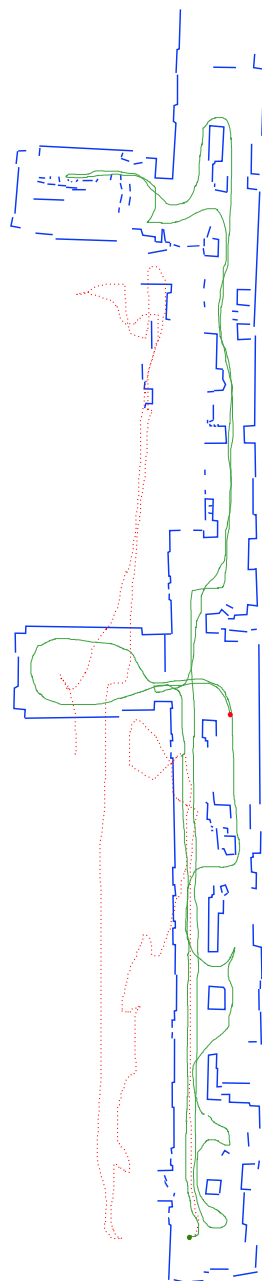


Fig. 5.11 Line map obtained by using clustering of hough parameters and EKF.

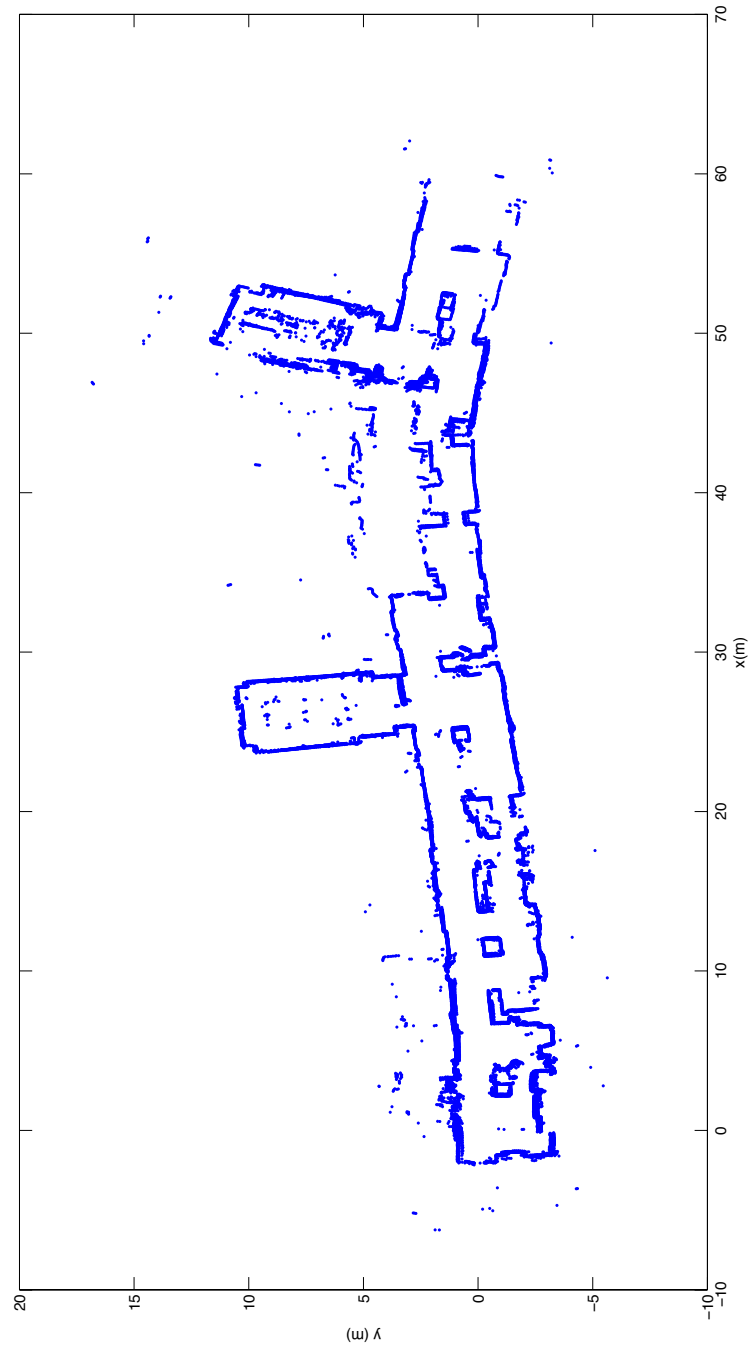


Fig. 5.12 ICP only generated point map. Over the long run the accuracy of the ICP map decreases over time.

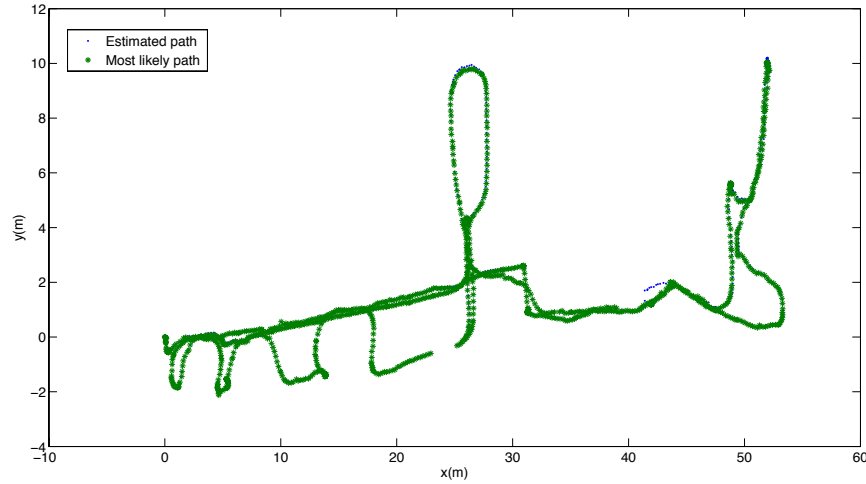


Fig. 5.13 Estimated path from the robot with most likely path.

applied in smaller maps, the vehicle travelled distance decreases, allowing an easier loop closure, increasing the accuracy of the final feature map. This continuous corrections makes necessary to apply the EKF-Localization after obtained the feature map of the EKF-SLAM, aiming a more accuracy in the estimation of the vehicle pose. The map is divided into small maps and after applying the EKF-SLAM to each zone, allowing to obtain the respective feature maps. The mapping procedure can be applied individually to each smaller map. The map was corrected by ICP matching method. Figure 5.12 shows the map generated by only using the ICP method. The map clearly shows that the error accumulates over time to distort the map.

Figure 5.13 shows the estimated and predicted paths of the robot which are corrected using the ICP method. The time taken for the ICP algorithm over the time is shown in Figure 5.14 whereas the robot's orientation with time is given in Figure 5.15. Figure 5.16 shows the odometry change over time for the x and y for the robot.

The particle filter grid mapping result is shown in Figure 5.17, and shows the grid map of the environment, where the obstacles are denoted by black grids on

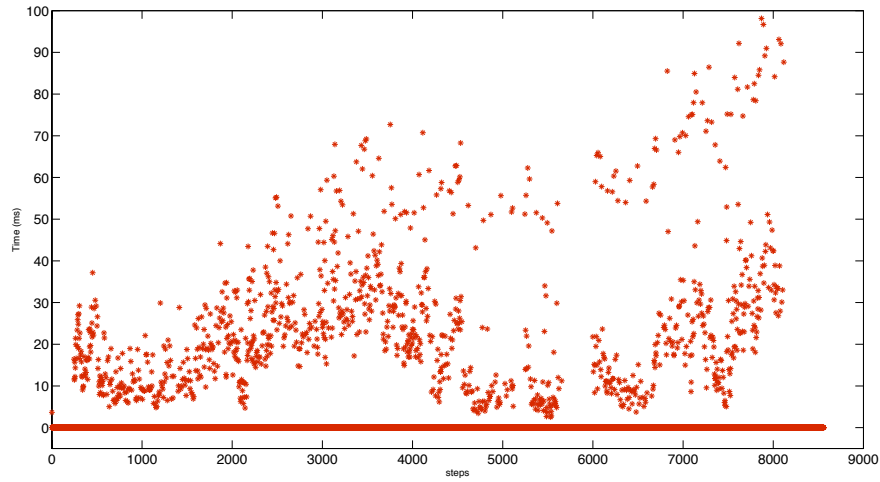


Fig. 5.14 Time taken by the ICP step.

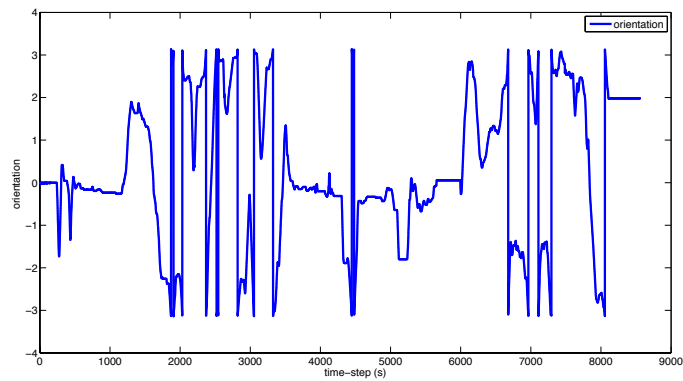


Fig. 5.15 Orientation of the robot over time.

the map and the empty areas are denoted by the white space. The number of particles were chosen as 200 for sampling. A comparison of the the RBPF-SLAM with the increasing number of landmark is given in Figure 5.19. It shows the

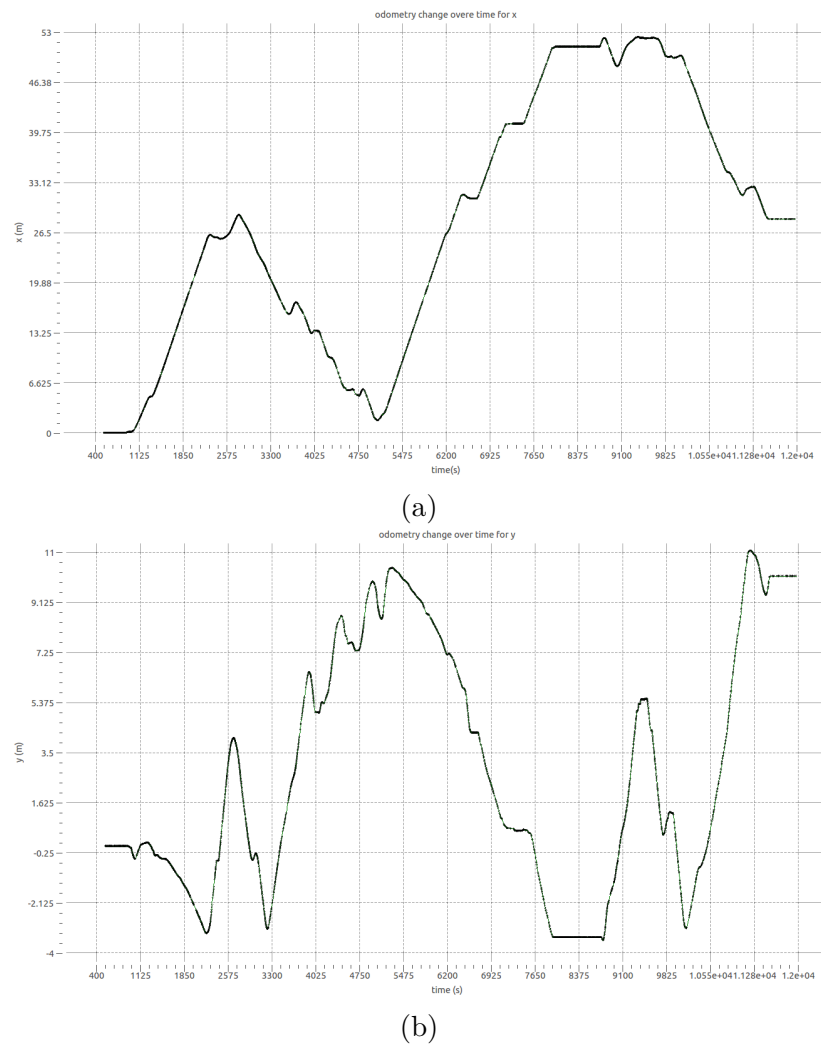


Fig. 5.16 Odometry over time for x and y for the experiment

error in position decreases with the increase in the number of particles. However it does come with the cost of increase in the memory cost.

Figure 5.20 from the GraphSLAM algorithm shows a much better map compared to other methods. The graph nodes are optimized using the Gauss Newton method and by utilizing the ICP in the front-end for local loop corrections and data association. The unoptimized graph of the trajectory is shown in Figure 5.21. The local loop closure constraints are indicated by the links in red color. These are the nodes where graph optimizes for the loop correction. The optimized graph of the trajectory are shown in Figure 5.22. The result shows the how the local loop closures are corrected with the trajectory. The resultant map is as shown in Figure 5.20. The unoptimized grid map of the same environment is shown in Figure 5.23. This map corresponds to the unoptimized graph from the Figure 5.21.

2. Scenario 2: The next environment that was chosen for the experiment consists of long straight corridors that run in loops. Two environments were chosen here, one with a large single loop. The total length of the environment is approximately 180 m. Another experiment was done by doing a double loop with a total length of 340 m. The mapping of such corridors is a challenging task, due to similar looking features that affect the data association process in the filtering. Also it is very easy for the robot to forget its position and to get lost in the map. When the robot is mapping such large areas and revisits any previously mapped region, due to the bounded errors from the odometry and scans, it becomes difficult to do correction. Such loop failures are a major problem in the SLAM. To solve this, only the graph SLAM algorithm was used. The robot was teleoperated first in a single loop and the odometry data and the LRF scans were stored. Figure 5.24 shows the result obtained from the RBP which could not correct the loop closure as can be seen. Figure 5.25 shows the result of the loop correction by the Graph SLAM algorithm. Because of its robustness the optimization process can correct small to large loop closure errors. For the double loop Figure 5.26 and Figure 5.27 shows the result of mapping by RBPF and graph SLAM respectively. The results shows the dexterity of the algorithm and is able to map large cyclic environments.

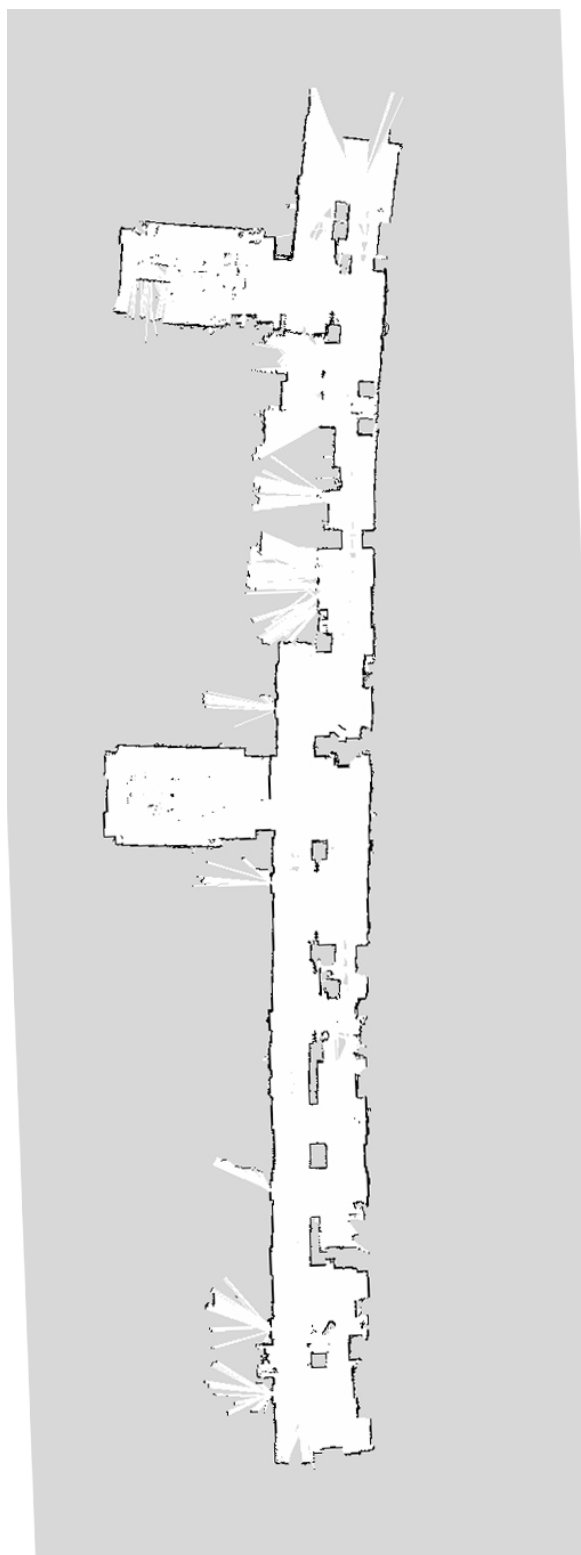


Fig. 5.17 Result from the particle filter. The map is consistent for most of the part except in the top where it bends a little.

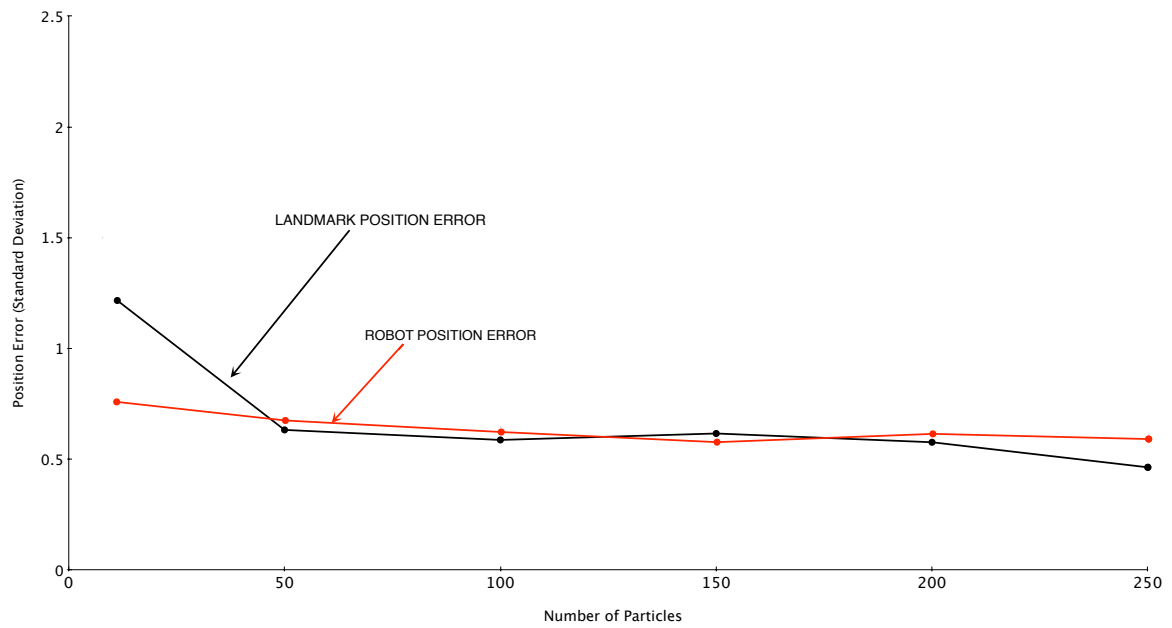


Fig. 5.18 Comparison of the FastSLAM algorithm with the increase in number of particles.

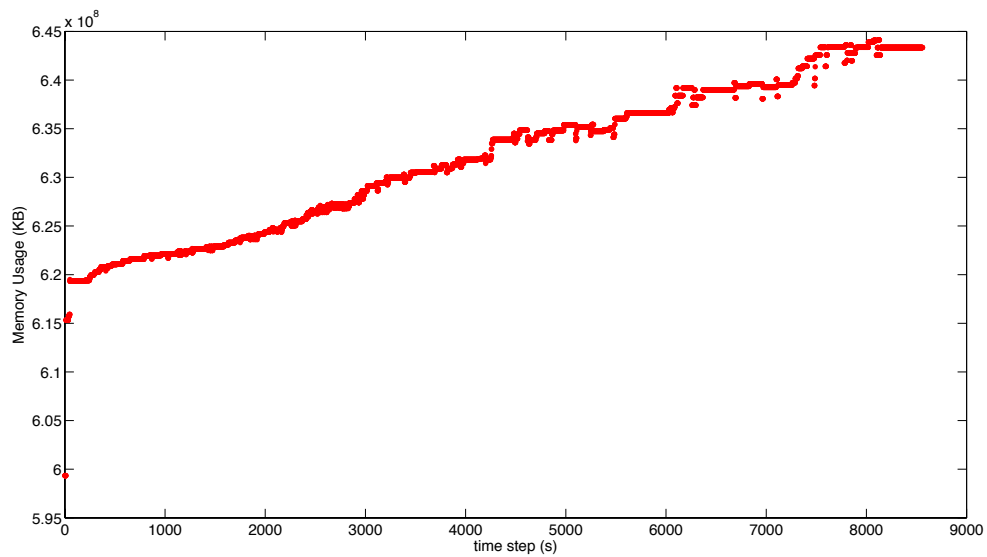


Fig. 5.19 Memory usage in case of 200 particles for the RBPF-SLAM.



Fig. 5.20 The pose graph grid map is very accurate to the real environment

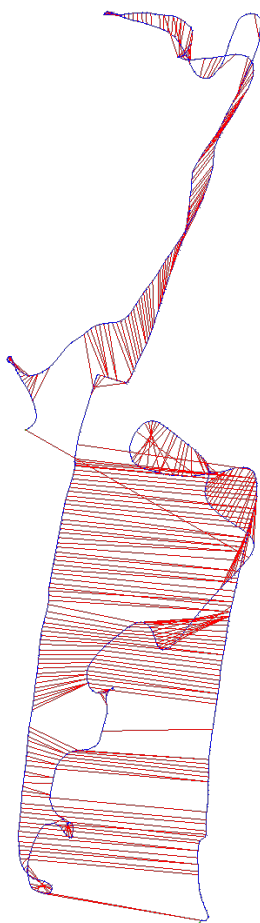


Fig. 5.21 Unoptimized pose constraints with links for the graph poses.

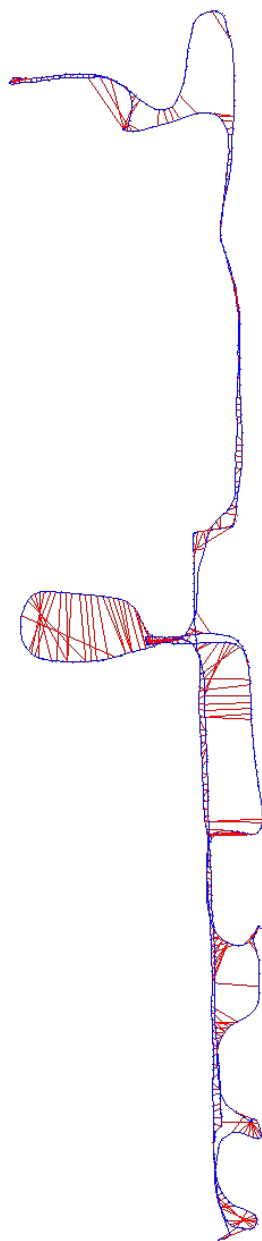


Fig. 5.22 The corrected and optimized poses with correct loop closure constraints.

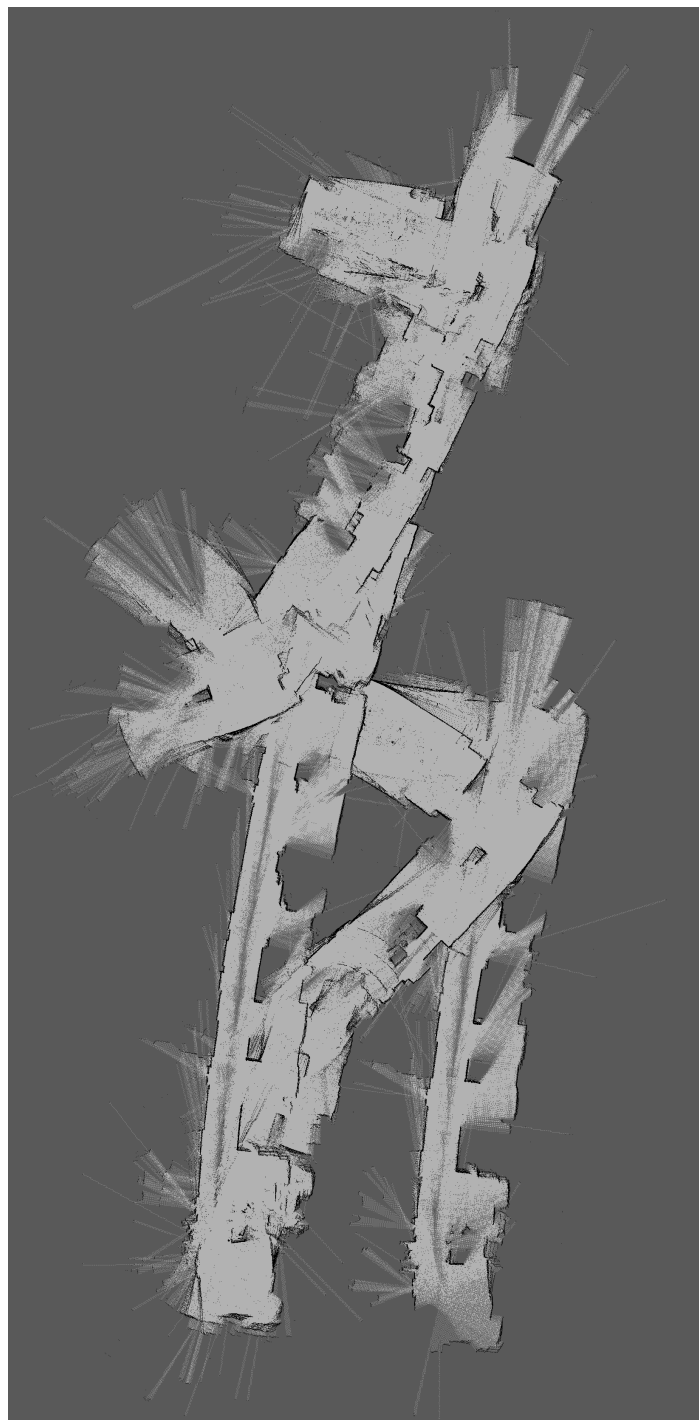


Fig. 5.23 Map from pure odometry

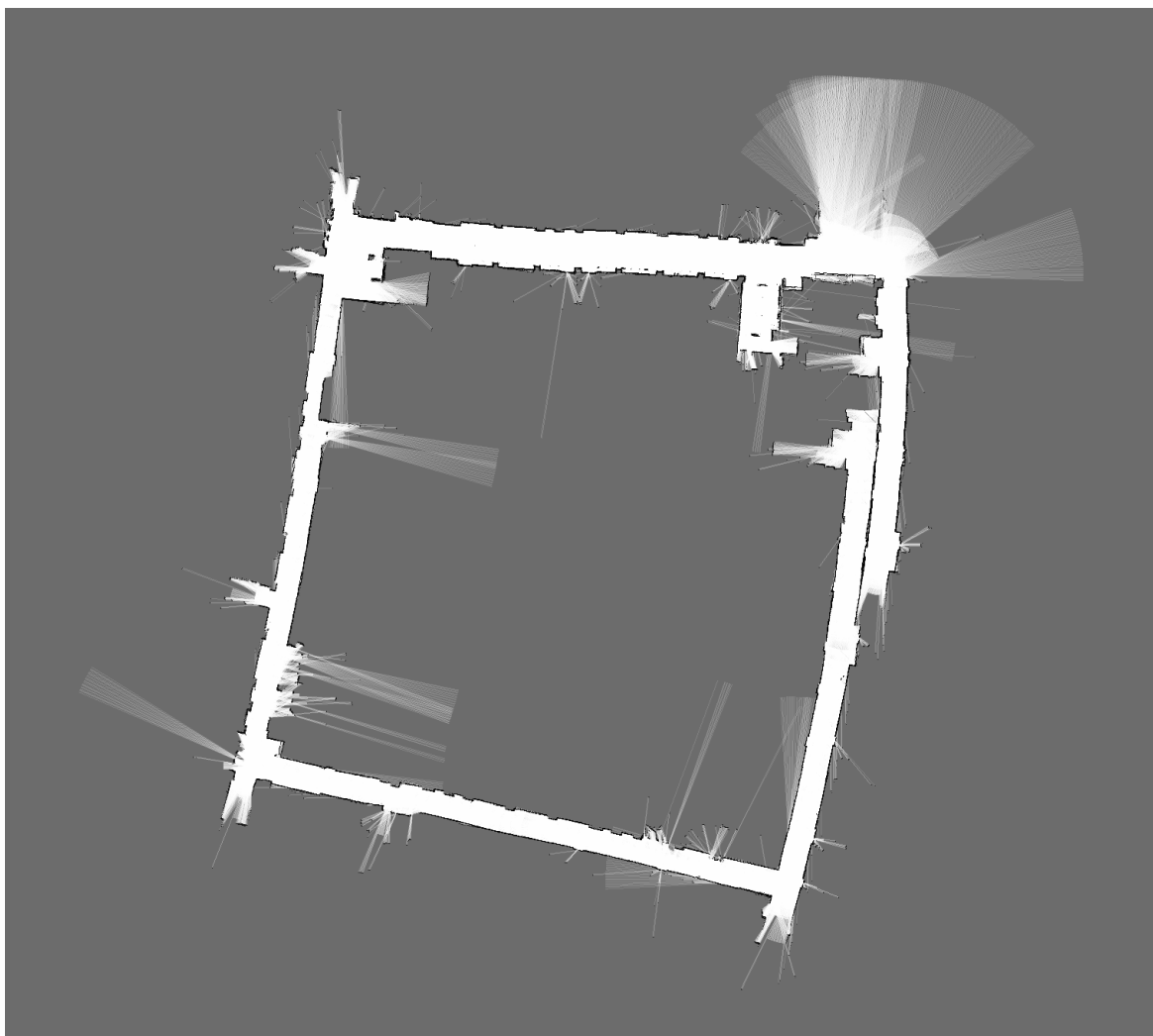


Fig. 5.24 Loop closure failure scenario for mapping a large loop. The scan matching fails when the robot revisits the same corridor. This attribute is due to similar looking corridors.

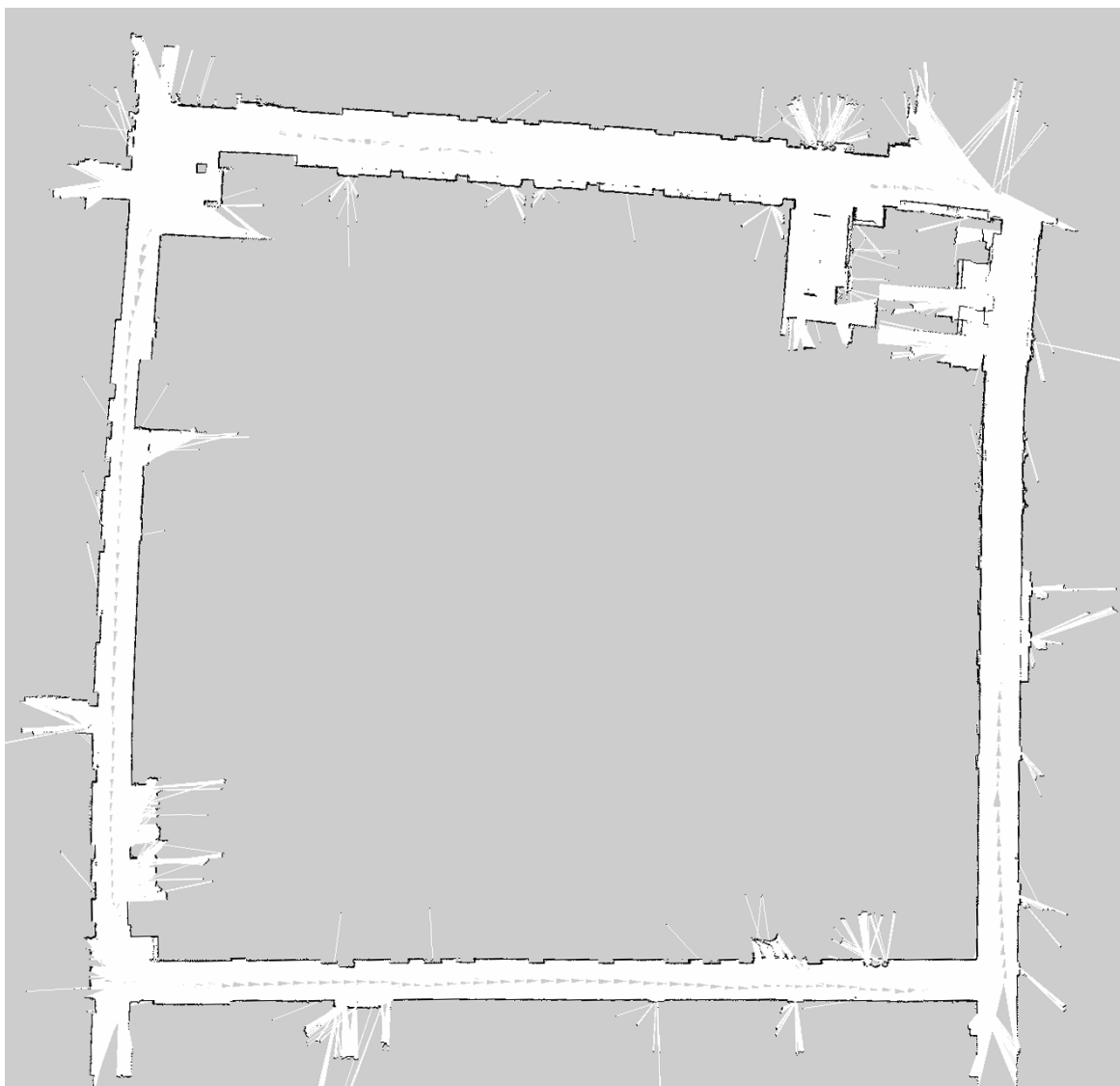


Fig. 5.25 Optimized loop closure constraint. The corrected loop is accurate to the real environment.

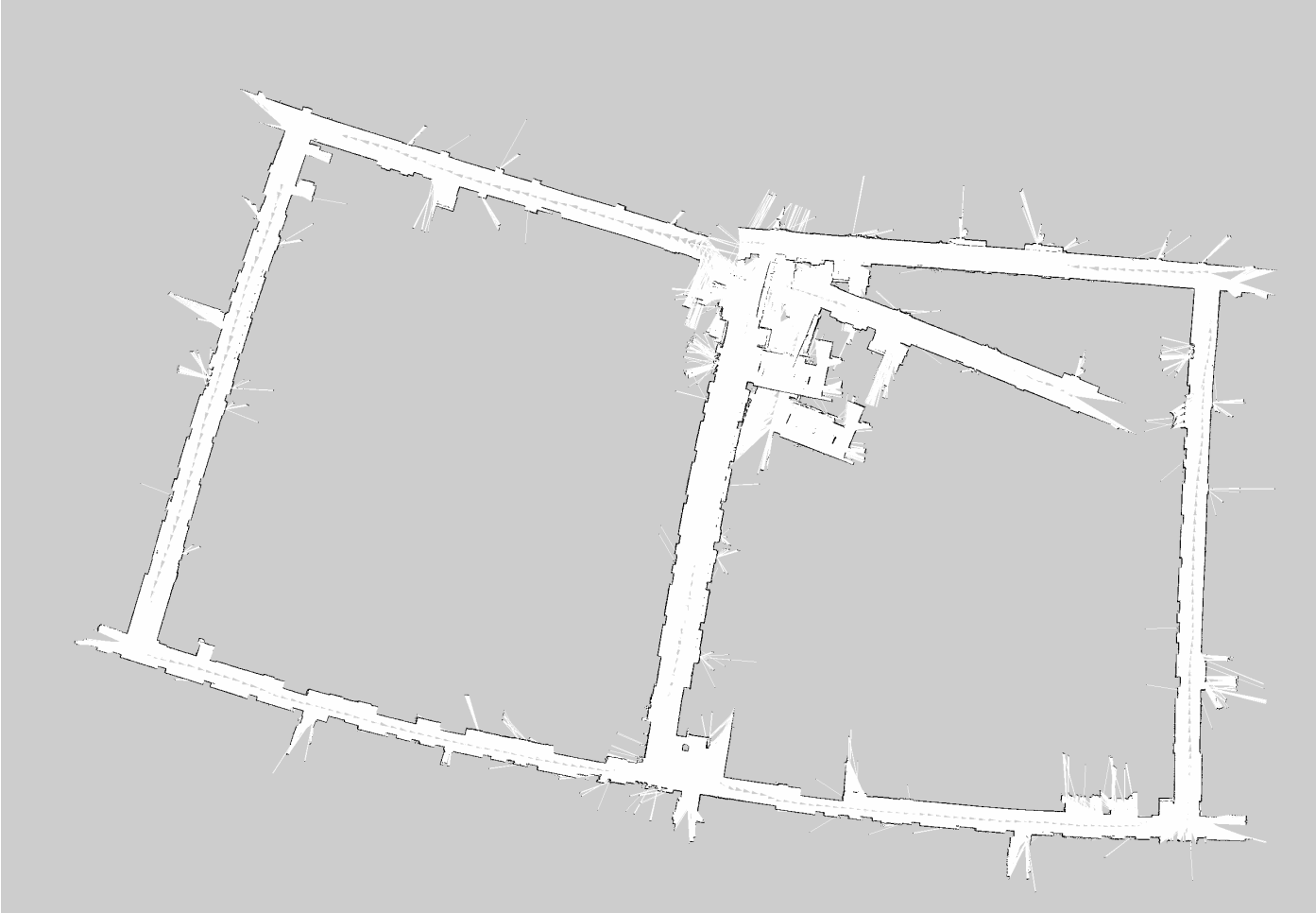


Fig. 5.26 Loop closure failure for long double loop map.

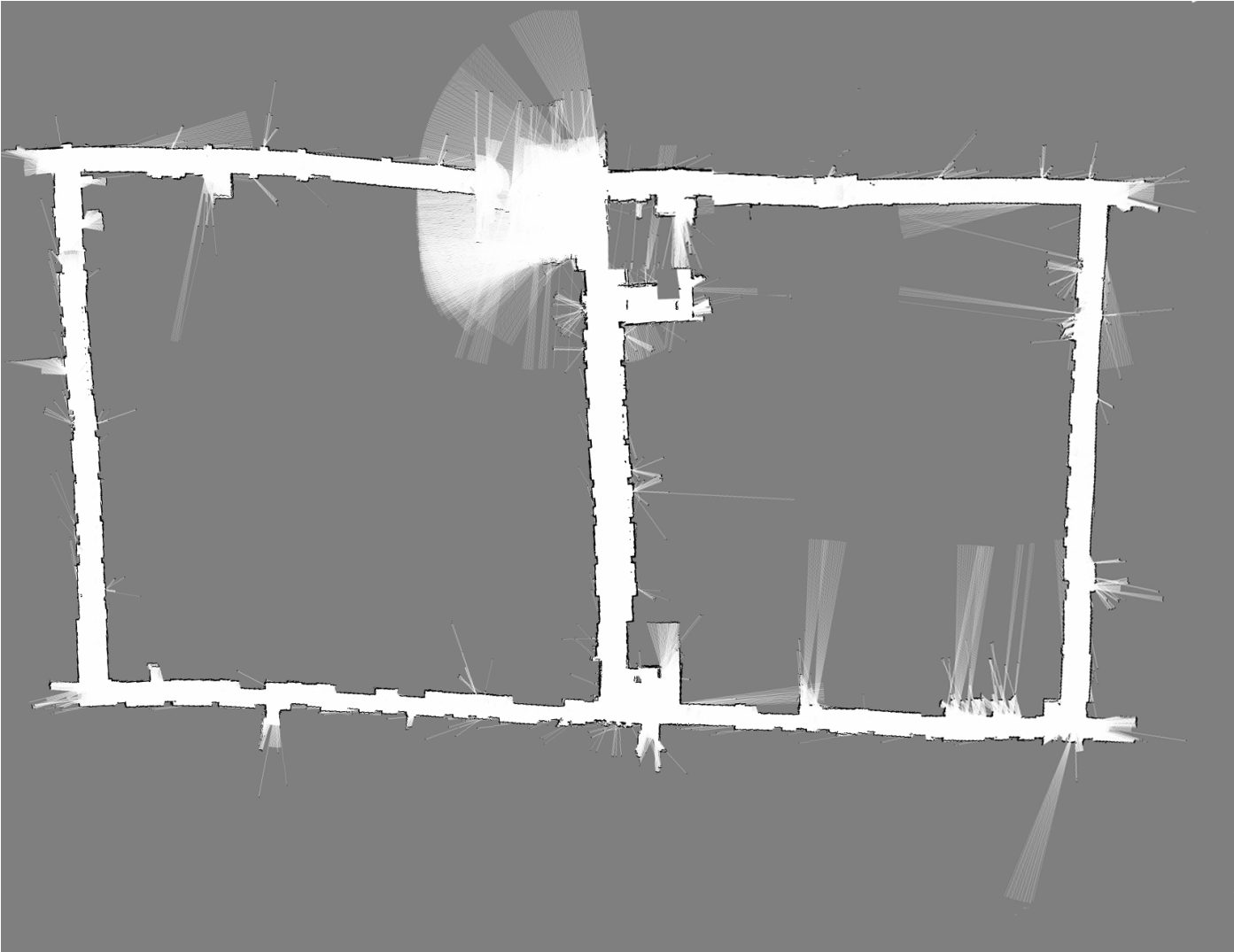


Fig. 5.27 Optimized loop closure constraint.

5.7 Conclusion

This chapter gave a brief overview to the implementation of the probabilistic filters that were discussed earlier. A detailed description of 2D line segment based SLAM using the EKF is presented. It is observed that the EKF tends to perform very slow when the number of features increases as more and more features are added to the algorithm. With a good odometry the EKF SLAM performs well and generally produces good maps when used with methods such as ICP for scan correction. Particle filter based SLAM on the other hand is just an extension of the EKF where each particle carries its own map and the particles are sampled and weighted to get the best particle. Although, RBPF particle filter is more robust and is fairly fast when mapping small to medium sized areas, it tends to get computationally heavy for a long and large mapping sessions. The reasons can be attributed to the sampling time that takes most of computation. To overcome this, another method, the Graph SLAM technique is presented which is much accurate and faster as it only depends on the poses and associated scans for the robot. The process uses optimization techniques that can be used in real time or offline for the sake of saving memory. The optimization process gets computationally expensive with the size of the map but can solve loop closure constraints pretty accurately. With ICP the graph SLAM produces accurate maps which are comparable to the particle filter based maps. Overall, both the particle filter and the Graph SLAM produced identical results but for large scale mapping the graph based SLAM is the better choice. EKF on the other hand is still a more popular algorithm as it does not depends on the size of the environment in comparison to the particle filter based approaches.

Chapter 6

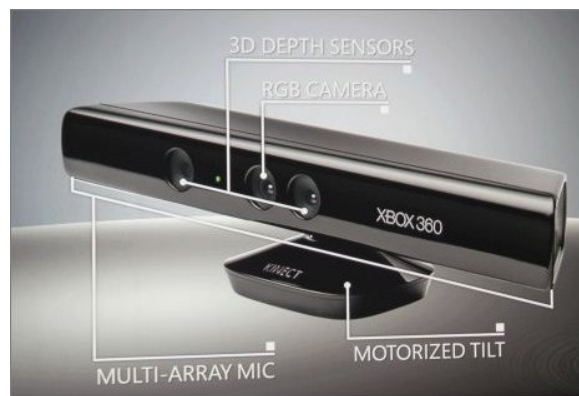
From 2D to 3D - Constructing Robust 3D Maps

6.1 Introduction

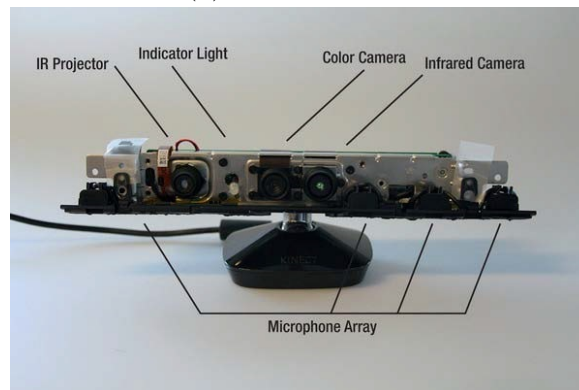
Most of the discussion so far in this thesis was towards constructing 2D maps in a structured environment using features (point features and line features). Although, the 2D maps provide enough information for the mobile robot to autonomously navigate and localize itself in the environment, the world in which the robots operate are not 2D but 3D. With one more dimensional information the robots can do very complex tasks of navigating in the 3D world, for eg. recognizing objects, perceiving dynamic obstacles in real time, avoid collisions with objects that cannot be avoided using a simple horizontal LRF (table legs, chairs, overhanging surfaces, slopes). As most robots operate in a more or less cluttered environment sensing the world in 3D makes sense. However, this comes at a cost of increased computation and complex algorithms for estimation. But nonetheless, its a very interesting problem and creates a new dimension for autonomous mobile robot applications. An earliest work in the visual SLAM was done by Davison et. al [20]. A number of new researches in this area has emerged in the last few years which deals with the visual navigation of robots [11, 15, 18, 46, 53]. A brief survey on visual navigation techniques is done in [29, 36]. This chapter briefly introduces techniques of constructing robust 3D maps that are built upon vision algorithms and knowledge from the previous chapters especially the Graph SLAM algorithm.

6.2 Microsoft Kinect

Most of the 3D mapping in this thesis is done using the Kinect sensor developed by PrimeSense. Initially the sensor was released for the Xbox 360 gaming console from Microsoft Corporation in 2010 but quickly became popular among the research community because of its low cost and enormous application to different fields of technology specially robotics. The Kinect sensor is composed by an RGB camera, 3D depth sensor, a multi-array microphone and a motorized tilt. In this work only the RGB and depth sensors are used to provide the input data.



(a) Kinect sensor.



(b) Internals of the Kinect sensor.

Fig. 6.1 The Kinect sensor from Microsoft Corporation.

The main characteristics of the Kinect sensor are [45]

- The RGB sensor is a regular camera that streams video with 8 bits for every color channel, giving a 24-bit color depth. Its color filter array is a Bayer filter mosaic. The color resolution is 640x480 pixels with a maximal frame rate of 30 Hz.

- The depth sensing system is composed by an IR emitter projecting structured light, which is captured by the CMOS image sensor, and decoded to produce the depth image of the scene. Its range is specified to be between 0.7 and 6 meters, although the best results are obtained from 1.2 to 3.5 meters. Its data output has 12-bit depth. The depth sensor resolution is 320x240 pixels with a rate of 30 Hz.
- The field of view is 57 deg horizontal, 43 deg vertical, with a tilt range of ± 27 deg.

The drivers used are those developed by the OpenNI (Open Natural Interface). The OpenNI framework offers high level functions which are mainly oriented for the gaming experience, such as gesture recognition and motion tracking. It aims to provide an abstraction layer with a generic interface to the hardware devices. In the case of the Kinect, one advantage is that the calibration of the RGB sensor with respect to the IR sensor is ensured, so the resulting RGB and depth data are correctly mapped with respect to a unique viewpoint.

Another depth sensor that is popular is the one developed by ASUS called the Xtion pro live sensor. The ASUS xtion pro live sensor is very similar to the Kinect sensor with few advantages of it being USB powered and having a better camera resolution. Both the sensors are used in this work.

6.3 Feature Extraction

There are many techniques for detection of keypoints or point of interest regions on objects or surfaces in computer vision. In this section, some techniques of feature extraction from the images are discussed. It is important that during the extraction and tracking of features from the images, such features need to be invariant to scale and rotation during the camera motion and robot motion. There are many methods in computer vision that solve this problem [19]. The main aspects for the feature are *Detection* of keypoint or the region of interest and *Descriptor*, which gives the characteristics of the region. which identifies an area of interest, and its descriptor, which characterizes its region. Typically, the detector identifies a region containing a strong variation of intensity such as an edge or a corner, and its center is designed as a keypoint. The descriptor is generally computed by measuring the main orientations of the surrounding points, leading to a multidimensional feature vector which identifies the given keypoint. Given a set of features, a matching can then be performed in order

to associate some pairs of keypoints between a couple of frames. Two of the most widely used feature is used in this work.

6.3.1 SIFT feature

SIFT or Scale Invariant Feature Transform by David Lowe [60] is the most widely used technique of feature detection in robotics and computer vision. The SIFT is an algorithm to detect distinctive, invariant image feature points which can be easily matched between two images for the purpose of image detection and recognition or to compute the geometrical transformation between the images. The main idea of the SIFT method is to define a cascade of operations following an increasing complexity, so that the most expensive operations are only performed to the most probable candidates. The steps are

- Finding the pyramid of Difference of Gaussian (DoG) in order to be invariant to scale and orientation.
- From this, stable keypoints are determined with a more accurate model.
- The image gradient directions is then used to assign one or more orientations to the keypoints.
- The local image gradients are then transformed to be stable against distortion and changes in illumination.

Detector When the robot is moving with the camera, the detected keypoints will get large and small, and it is important that such keypoints are invariant to such motions of rotation and scale. The SIFT algorithm can handle this effectively. The scale space is defined as the convolution of an image I with Gaussian G and variance σ

$$L(x, y, \sigma) = G(x, y, \sigma)I(x, y) \quad (6.1)$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2 + y^2)}{2\sigma^2}} \quad (6.2)$$

The DoG is the difference between two layers in scale space along σ axis.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma)) - G(x, y, \sigma)I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \quad (6.3)$$

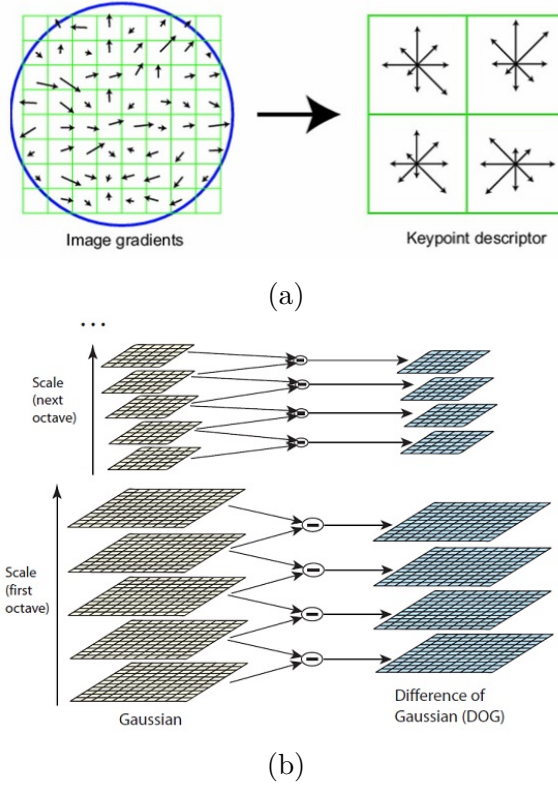


Fig. 6.2 Difference of Gaussians at different scales and SIFT descriptor

This gives a close approximation to the scale normalized Laplacian of Gaussian $\sigma^2 \nabla^2 G$ as shown in [45, 59]

$$\sigma^2 \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \quad (6.4)$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G \quad (6.5)$$

The factor σ^2 allows the scale invariance. Then, the localization of the keypoints is done by finding the extrema in the DoG, which approximates the Laplacian $\sigma^2 \nabla^2 G$.

Descriptor In order to provide a good basis for the matching, the descriptor has to be highly distinctive and invariant to remaining variations such as change in illumination or 3D viewpoint. The SIFT descriptor is based on a histogram of local oriented gradients (HoG) around the keypoint. It is stored as a 128 dimensional feature vector (4x4 descriptor with 8 orientation bins). The descriptor has to be distinctive and invariant to variations for illumination and 3D view. The Figure 6.2 illustrates this concept with a 2x2 descriptor.

Matching Once the keypoints are found and the descriptors computed, the next step is to perform fast searches on the features in order to identify candidate matching, to associate some pairs of features between the two sources[60].

6.3.2 SURF feature

The Speeded Up Robust Feature (SURF) is inspired by the SIFT descriptor, both using local histograms [8]. The main difference is the performance, lowering the computational time through efficient use of integral images for the image convolutions, Hessian matrix based detector (optimized through approximations of the second order Gaussian partial derivatives) and sums of approximated 2D Haar wavelet responses for the descriptor. It provides robust detector and descriptor to be used in computer vision for the tasks like object recognition and reconstruction [9]. The standard version of the SURF is several times faster than SIFT and is more robust against different image transformation.

The SURF process is significantly faster than SIFT and gives significantly better results. Nevertheless, tests were conducted using both the features depending on the environment where the experiments are performed.

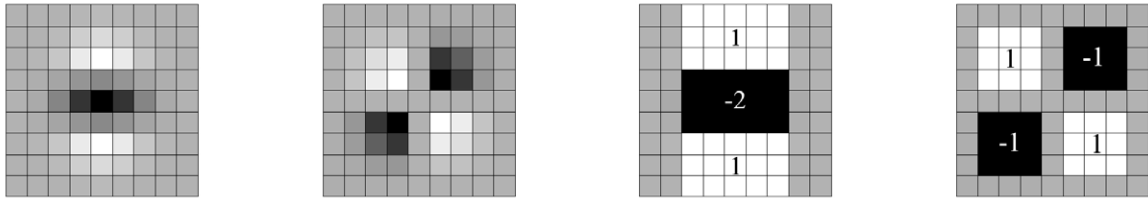


Fig. 6.3 SURF detector and approximation of Gaussian second derivative

6.3.3 Initial Matching

Once the features are computed, a matching between couple of frames is computed. A nearest neighbor search, or the Euclidean distance is used to compute on the descriptors of these features. The SIFT Library proposes an implementation of the Best-Bin-Search described by Lowe [60], based on kd-tree. For each feature in the target frame, the two nearest neighbors are searched in the source frame. If these neighbors are close enough (with respect to a fixed threshold), then the result is considered to be valid. This initial search is done on the 2D features. For robustness, matches are rejected for those keypoints where the ratio of the nearest neighbor distance to the second nearest neighbor distance is greater than a given threshold (eg. 0.5). After this,

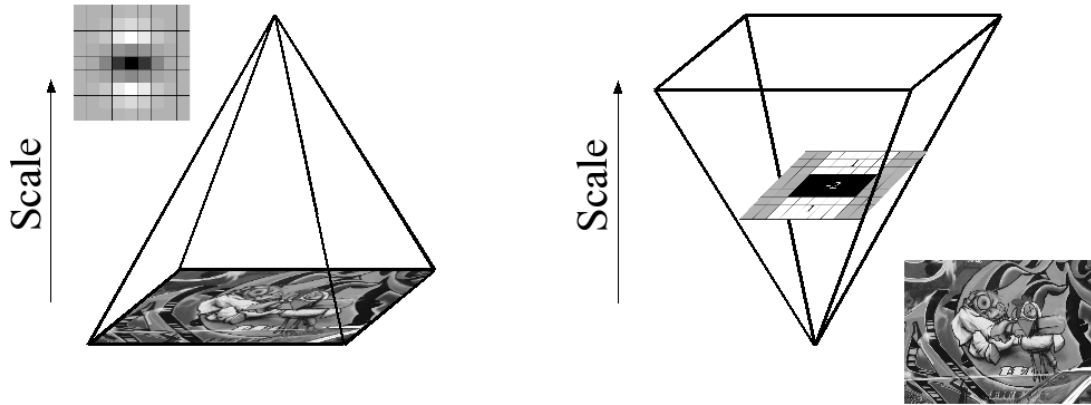


Fig. 6.4 SURF scale space built..

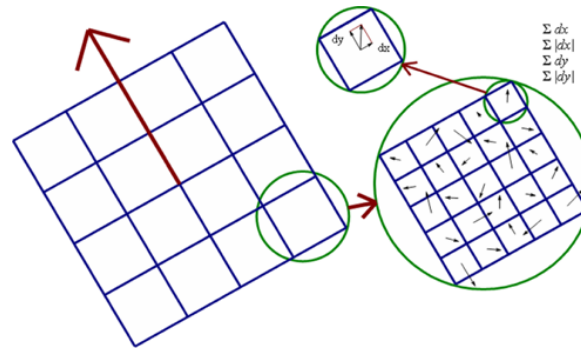


Fig. 6.5 SURF descriptor

the depth information is used to keep the features that are close enough to the camera, the depth information is not considered reliable enough above a given range (around 5-6 meters). Figure 6.6 shows correct matching between two images of the same scene from the robot camera by the SURF features. Figure 6.7 shows the discarded features because of not enough inliers available for the matching.

6.3.4 RANSAC

The RANdom SAMple Consensus (RANSAC) is an iterative algorithm, widely used in computer vision to estimate the parameters of transformation give a dataset [33]. By using the least square technique the parameters that satisfy the data set can be found, but this does not give an optimal solution when there are noise in the data. Hence, the idea is to find the parameters which are valid for most of the points, by discarding the noisy points. For each iteration, small number of samples are randomly selected to define a model, representing one hypothesis. Then, this model is gradually

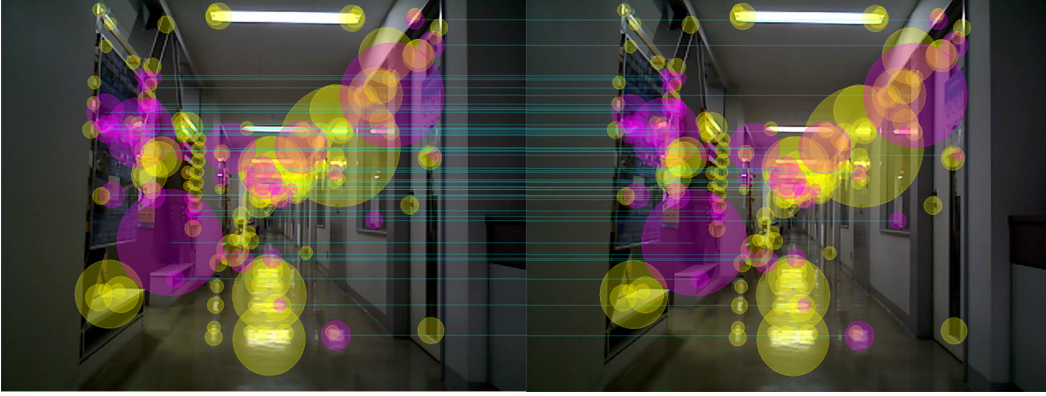


Fig. 6.6 An example showing the matching of SURF features for a corridor environment

evaluated for the entire dataset along with an error function. This process is iterated by selecting new samples for each new iteration and the keeping the best transformation that is found. After several steps, the algorithm converges to a better transformation with a lower error. However, this does not necessarily gives the best estimate as all the possibilities are not tested. The general algorithm is presented in Algorithm 4. Figure 6.8 describe the RANSAC process on a set of 2D points. First the process randomly selects points (here $k = 2$). From these points a model is defined. A line is then drawn between the two chosen points with an area of validity defined by a given threshold. The model is then evaluated by measuring the error for each point by computing the distance to the line. This separates the dataset into two subsets. The ratio of the inliers relatively to the total number of items can be given a numerical evaluation of the model. The initial model is generally recomputed taken all the inliers into account.

The number of iterations are chosen by defining the ideal value from the desired probability. Considering a sample of k points, randomly chosen, if p denotes the probability that at least one of the sample set does not include an outlier, u is the probability of observing an inlier, then

$$1 - p = (1 - u^k)^N \quad (6.6)$$

Also, $v = 1 - u$ is the probability of observing an outlier. The number of iterations are therefor given as [45]

$$N = \frac{\log(1 - p)}{\log(1 - (1 - v)^k)} \quad (6.7)$$



Fig. 6.7 Wrong feature matching: Wrong feature detected during the matching and are ignored.

The value of threshold generally depend on the problem and applications and there are many ways to solve this problem [100].

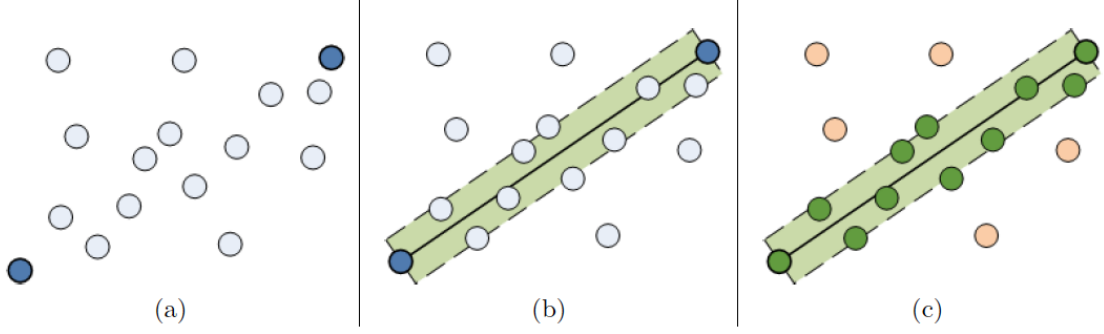


Fig. 6.8 Illustration of RANSAC process. The green points are *Inliers* and orange points are the *Outliers*

The RANSAC is used along with the ICP algorithm to get an initial guess for the ICP procedure.

6.4 Feature Transform

The next task is to track the features during the motion of the camera. This is mainly done by associating them between different frames. Selected keypoints are matched for a couple of frames and the process is repeated for the whole sequence of frames. As the same features are not detected for every frame due to the motion of the camera and noises present, the association is not so accurate. The best matching consists of finding the correct association with a good belief. The matches are then fit to the model. The matches that are fit are called the *inliers*, while the mismatched and discarded matches are called the *outliers*.

6.4.1 3D Transformation Estimation

In the next step, a rigid transformation between the two sets of points is calculated. In this operation each feature point in the source frame is projected to the set of points in the target frame. This transformation is a perspective projection for 6 degrees of freedom, composed by a rotation matrix and a translation vector in 3 dimensions. This transformation can be written by using a 4×4 matrix and homogeneous coordinates. We can then project a point P where $P = (x, y, z, 1)^T$ simply by applying this transformation matrix to the point:

$$P' = T_{transform}P \quad (6.8)$$

The points that are defined by the initial matching pairs then needs to be converted in 3 dimensions. To make the next steps easier and avoid further conversions, the same

Algorithm 4: RANSAC Algorithm

Data: Dataset of points, ($bestModel, bestInliers \leftarrow \phi$)

```

1 Define number of iterations  $N$ ;
2 for  $iteration = 1$  to  $N$  do
3    $samples \leftarrow$  pickup  $k$  points randomly;
4   compute  $currentModel$  from  $samples$ ;
5    $inliers \leftarrow \phi$ ;
6   for all points do
7     Evaluate  $currentModel$  for the point and compute its error;
8     if  $error < threshold$  then
9        $inliers \leftarrow inliers + point$ ;
10    end
11  end
12  Count number of inliers and compute mean error;
13  if  $currentModel$  is valid then
14    |
15  end
16  Recompute  $currentModel$  from  $inliers$ ;
17  if  $currentModel$  better than  $bestModel$  then ;
18     $bestModel \leftarrow currentModel$ ;
19     $bestInliers \leftarrow inliers$ ;
20 end
21 return  $bestModel, bestInliers$ 

```

coordinate system is used for all the work, defined as follows:

$$\left\{ \begin{array}{l} x : \text{depth, positive forward} \\ y : \text{height, positive upwards} \\ z : \text{width, positive to the right} \end{array} \right. \quad (6.9)$$

From the depth value and the screen coordinates (with RGB color and a resolution of 640×480 pixels), it is possible to compute the points coordinates in 3D. Let (u, v) be the point coordinates in pixels, we have then:

$$P(x, y, z) \left\{ \begin{array}{l} x = \text{depth} \\ y = y = -\left(v - \frac{480}{2}\right) \times \frac{\text{depth}}{\text{focal length}} \\ z = \left(u - \frac{640}{2}\right) \times \frac{\text{depth}}{\text{focal length}} \end{array} \right. \quad (6.10)$$

Considering the initial matches, the next step is then to find a transformation matrix, that gives a satisfying projection for most of these points. Ideally, we could simply compute a transformation by a least square method for all these points, each matching pair defining an equation. The minimum number of pairs required for this would be three, and the more points, the better defined the system would be. However, because of the uncertainty due to the sensory noise and the feature matching by itself, a single point which location is incorrectly estimated could lead to a significant error in the final transformation. Therefore, a better transformation can be found iteratively with the RANSAC method as described earlier for the 3D case. The value of n was chosen as $N = 20$. Generally, if the quantity of features is high, the probability of having an outlier is lower at this step, as most of the mismatches have already been excluded by the initial matching.

To compute a transformation (an hypothesis), only the k samples from the initial matches are used each time. The 3D points are converted to homogeneous coordinates and the transformation is given by solving the corresponding equation from the known constraints which are defined by the chosen points. This can be done through a Singular Value Decomposition (SVD) of the covariance data. To do this, we use the minimal number of points to solve this equation, $k = 3$ [45]. To evaluate a transformation, each sample taken from the initial matches (source) is then projected according to this transformation. A 3D vector is computed from the difference between the projected point and the real point taken from the matching point (destination). The error is the norm of this vector.

6.5 Pose Graph

To solve the 3D SLAM problem, it can be defined as looking for the best estimation of the robot/camera poses, by reducing the uncertainty due to the noise affecting the measurements. From the previous discussion, the belief state can be extended to 3D, with x_k denoting the state of the robot at time k , z_k is the measurement (camera images or laser scans) and u_k is the control data. Different filtering techniques like the EKF or the particle filter can be then applied to calculate the best estimate of the robot poses or camera over the period of time. However, these filtering approaches grow exponentially with number of features and increases the computation cost.

Therefore, the graph based approach is utilized in this study which can work very efficiently offline by triggering an optimization process in the backend to handle large features [95]. Here the graph is called the pose graph, where the nodes (or vertices)

represent the state variable describing the poses of cameras, and the edges represent the related constraints between the observations connecting a pair of nodes. In the later stages, the optimizer is run to find the most likely configuration of the nodes.

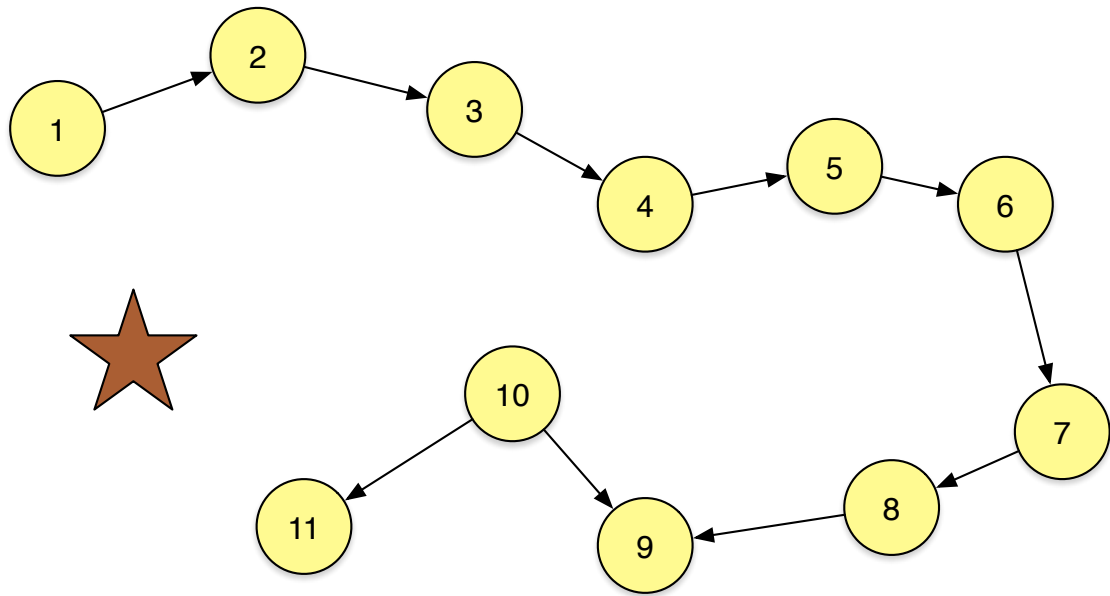
As discussed in earlier chapters, the graph SLAM approach only maintains the poses of the robot or camera state over the period of time with sensor measurements (images and laser scans). In case of 2D, these were then utilized for the building an accurate map. The same principle can be extended to the 3D case while maintaining all the point cloud data for all poses in the graph. Where the nodes are connected using edges and links.

6.5.1 Loop Closure

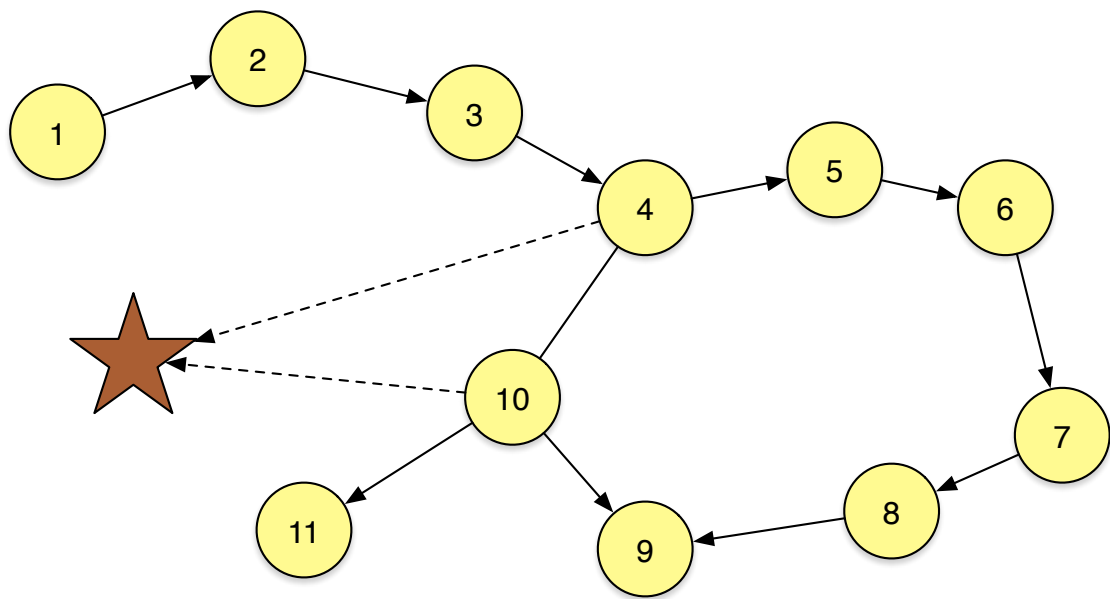
Loop closure problem can be solved using the pose graph approach. Without making an assumption on the path followed by the robot, and simply by keeping the history of the past frames, it is possible to check if the current frame matches with some of the previous ones. If the current observation contains enough similarities with another observation seen in the past, a transformation can be computed between these frames and a new constraint can be inserted from it. This can be repeated several times. With these new constraints, the cumulated error of the graph can then be considerably reduced. Theoretically, all the poses could be linked together this way, if such a relation exists. The optimization of the graph consists in finding the optimal configuration of vertices to respect all the given constraints. Figure 6.9 shows the loop closure detection method in pose graph. A landmark already seen in the past can trigger a link between the two poses that were previously separated in the sequence. The detection of the loop closure leads to new constraints that can be inserted into the graph as new edges, linking the related poses. The optimization process is shown in Figure 6.10. First the initial poses are constructed in the graph. In the next stage, loop closure links are found with edges and links with the initial loop closures. The graph is then optimized which compensates all the poses to the new poses after the optimization process.

6.5.2 Pose Estimation

From the known initial pose, the first step is to determine the estimation of any pose after the succession of transformation. Consider a finite sequence of frames from 0 to N , with P_k be the pose. For 6DOF, composed by 3 axis of rotation and translation, it can be represented by a 4×4 matrix with homogeneous coordinates, where R is a



(a)



(b)

Fig. 6.9 Loop closure detection

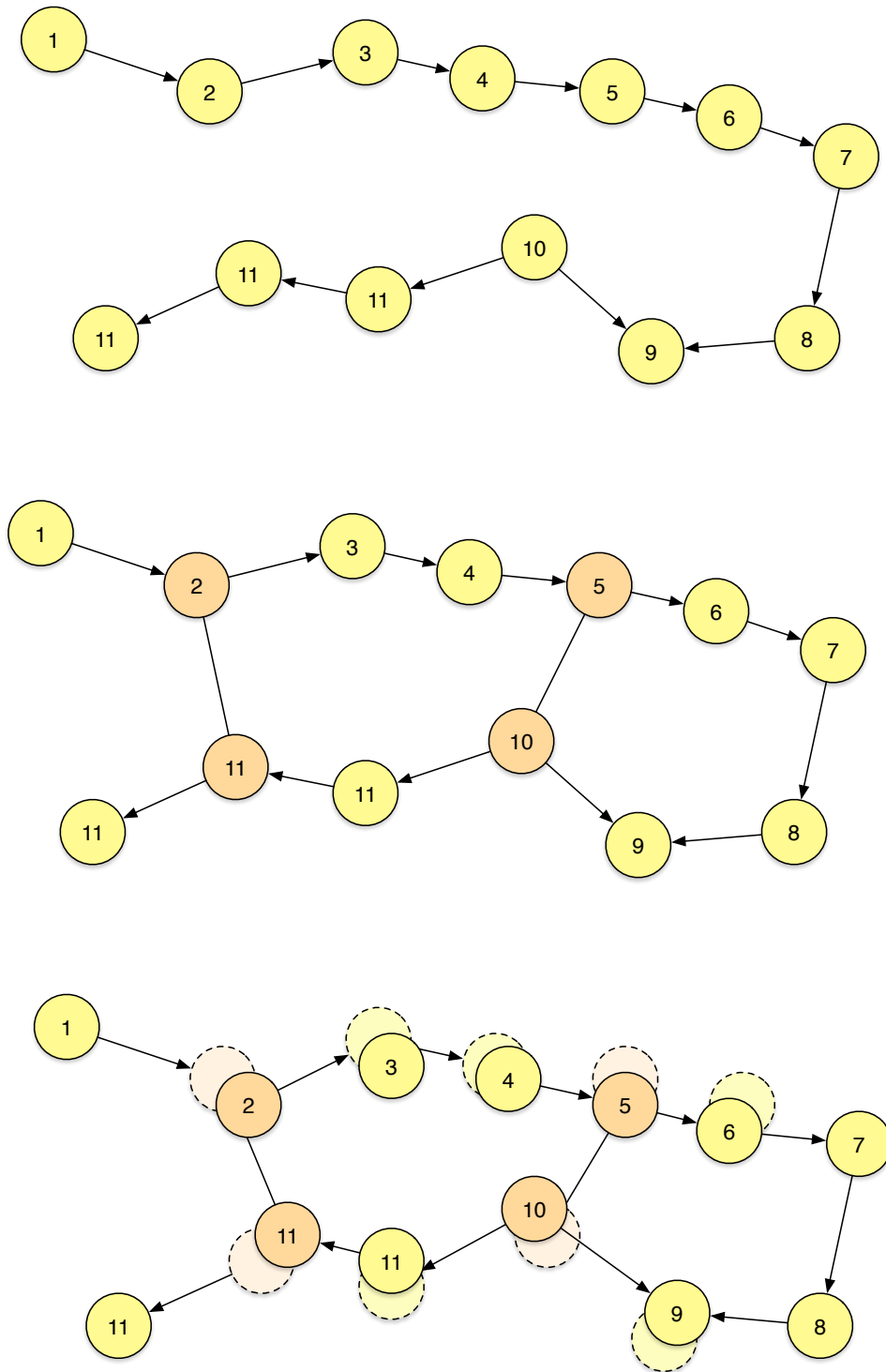


Fig. 6.10 Graph optimization process

3×3 rotation matrix and t , a translation vector [45]:

$$P_k = \begin{bmatrix} & & t_x \\ & R & t_y \\ & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.11)$$

For $i > 0$, the transformation T_{i-1}^i binds the position P_{i-1} to the position P_i . Each transformation T can also be represented by a similar 4×4 matrix. If P_0 determines the initial position, we can then compute the position P_i by combining all the transformations like:

$$P_k = \prod_{i=k}^1 T_{i-1}^1 P_0 \quad (6.12)$$

For example, for pose P_4 it is given by

$$P_4 = T_3^4 T_2^3 T_1^2 T_0^1 P_0 \quad (6.13)$$

The initial position is defined by the initial orientation of the camera for each axis and stored in the initial rotation matrix R_0 , and the initial vector $t_0 = (x_0, y_0, z_0)^T$ by

$$P_0 = \begin{bmatrix} & & x_0 \\ & R_0 & y_0 \\ & & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.14)$$

We can also define the initial position to be at the origin of the coordinate system, which is generally followed for any SLAM system for initialization

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.15)$$

Equation (6.12) gives an estimation of the pose for the graph. The new poses are inserted only after a significant change from a given pose called as *keyposes*. A simple solution is to define a keypose only after having achieved a move, with a given amplitude defined by a threshold. By cumulating the unitary moves between each

couple of frames in the sequence, the creation of a new key pose is triggered once a given distance in translation or a given angle in rotation has been reached, with respect to the last key pose. The thresholds are set by parameters, here 0.1m for the distance and 15° for the rotation (both values consider the variations on the 3 axis cumulated together). Another solution would be to compare the points of interests between the frames and trigger the creation of a new keypose when the number of points in common falls below a given threshold, but it is not given that the absolute number of points is representative enough to describe the amplitude of the move. For each keypose, a node is inserted in the graph with an edge linking the new keypose with the previous one. A standard edge represents the rigid transformation between the two keyposes. Between each keypose, they may be a more than a couple of frames. To compute the corresponding transformation, the resulting matrix is found by multiplying all the intermediate matrices for each couple of frames.

For loop closure the goal is to select the past frames that are more likely to match with the current frame, giving a higher priority to the oldest frames. This could be done defining a probability density function with a higher probability for the oldest frames. This probability could also be defined considering the knowledge of the context, in terms of features, or in terms of estimated position, preselecting the candidates with the highest likelihood. To improve the quality of the loop closure detection, once a candidate frame is found, the check is extended to its neighbors, looking for a better transformation. The ratio of inliers is used to measure the quality of the transformations and to compare them.

6.5.3 Optimization

Once the graph has been initialized with the poses and the constraints from the loop closures, it can be optimized. The method used here is Levenberg-Marquardt with a linear Cholmod solver, the details of which is beyond the scope of this thesis and can be found in [55, 96]. The optimization is done with a predefined number of steps. Once the graph has been optimized, the vertices are corrected and the new estimations of the camera poses can be extracted. Similarly to the insertion, the inverse operation is done to convert the information from the graph vertices to 4×4 matrices.

6.6 Map construction

Finally, once the camera poses are determined with a good belief, the reconstruction of the scene can be performed [95]. For each frame, a 3D point cloud can be generated

from the RGB and depth data, providing the colour information for each point. However, in order to reconstruct the whole scene, they have to be combined together from a unique point of view. This process is called the point cloud registration. Once the camera positions are known, each point cloud is transformed by projecting all its points according to the corresponding camera position, given by the Equation 6.12. This is done relatively to the first pose, which is predefined. The transformed point clouds are then concatenated together to build the scene [44].

6.7 Experiment Results

Experiments were conducted in indoor environments of varying nature. The environments are assumed to be static and no dynamic obstacles are present in the environment. For the purpose of comparison, the same two scenarios were chosen from the last chapter. The first environment is an indoor corridor like environment (A-3 wing of the Graduate school of Engineering). The robot was teleoperated using joystick inside the corridor passages, while the odometry messages, the LRF scans and images with point cloud data were collected. First, the graph SLAM algorithm was applied and a 2D map with the correct poses were obtained. As stated earlier the point clouds were then reconstructed with the corrected poses and assembled using the standard ICP method. The point clouds were assembled offline.

Figure 6.11 shows the final assembled 3D map from the corrected poses that were obtained after optimizing the poses. The map is overlaid over the laser scans and can be seen with the 3D map.

The statistical analysis is given in Figure 6.12. It shows the plots of scan-matching time taken during the algorithm across all the location indexes that are stored for the graph SLAM (Figure 6.12a), the odometry variance correction (Figure 6.12b), keypoint detection time for all the images that are used for searching the loop closure (SIFT) (Figure 6.12c) and the total time taken by the process (Figure 6.12d). The drop in the time is due to the stopping of robot during the experiment to avoid people that were walking in the corridor.

Figure 6.15 shows the result of mapping the long corridors (ground floor of the Graduate School of Engineering) in a loop. Figure 6.16 shows the point clouds with the RGB values. The unoptimized graph trajectory is shown in Figure 6.13 and the corrected poses are shown in Figure 6.14. Figure 6.17 shows the same 3D map overlaid on top of the actual building to compare the accuracy of the map.

6.8 Conclusion

This chapter presents a novel 3D mapping solution to SLAM using a multi-sensor fusion approach. Utilizing low cost depth sensors and LRF the proposed method uses visual features from the environment that are extracted using the SIFT features. LRF is used for local scan matching using standard ICP method for correcting local loop closures. The method uses the optimized graph nodes of all the poses that are corrected in the back end to get the corrected robot trajectory. This is generally useful in cases when only using the LRF is not feasible for correctly mapping the environment, for eg. when the robot cannot see obstacles that are not in the scanning area of the LRF or are too short to scan. Utilizing low cost depth sensors such obstacles can be effectively mapped from the area for efficient obstacle avoidance in a 3D world. The experiments performed in very cluttered and narrow environments produces high noise in the data, but using the multi-sensor fusion approach very accurate and efficient 3D maps can be built using the proposed methods. The application to mapping large scale cyclic maps can also be corrected and produces high quality maps for the robot navigation application. The 3D maps can be used for object manipulation and semantic mapping and it is the area for future research.

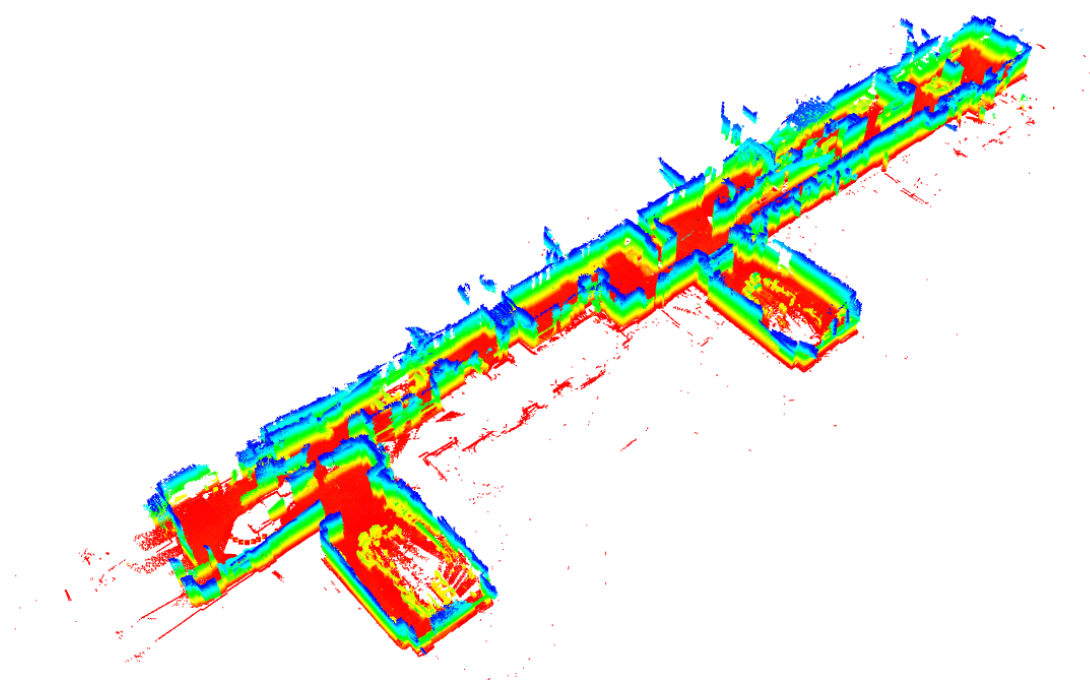


Fig. 6.11 3D map of assembled point cloud for Experiment 1.

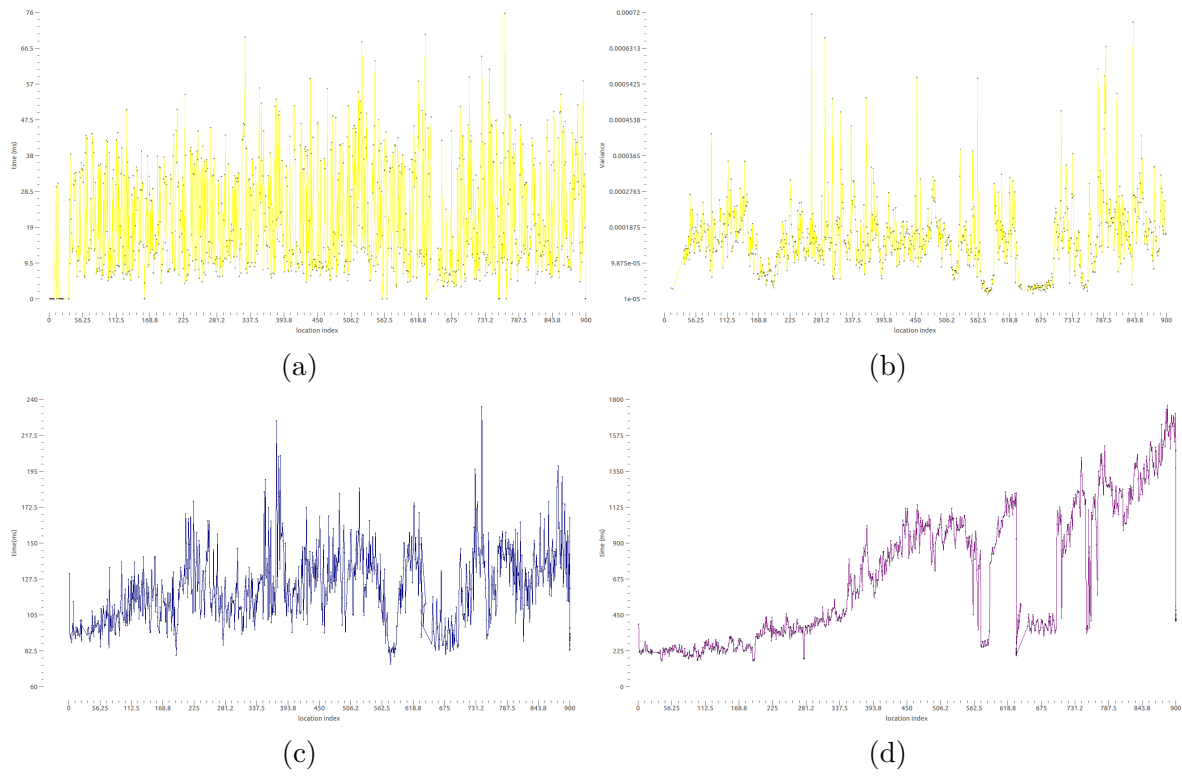


Fig. 6.12 Statistical analysis of Experiment 1. (a) Scan matching time for the LRF data over the location indexes. (b) The odometry correction variance over location indexes. (c) The time required for the keypoint detection from the images. (d) Total time taken for the algorithm.

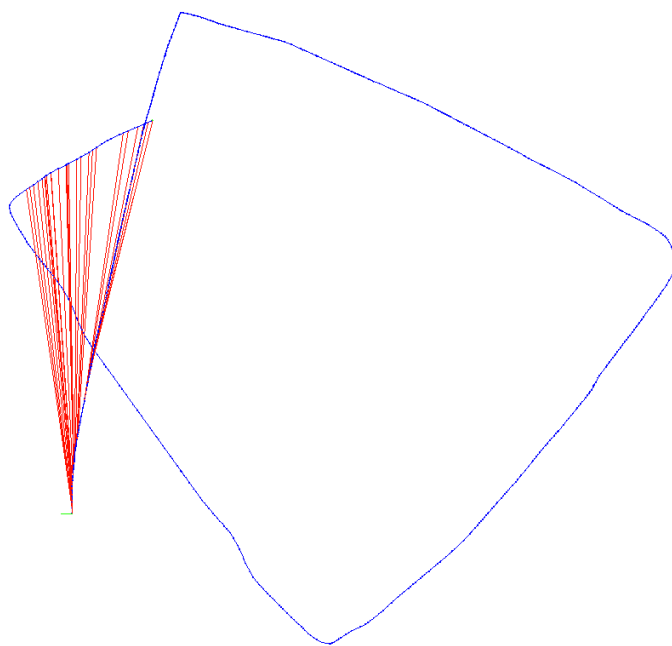


Fig. 6.13 Trajectory before optimization, the failed loop closure and link constraint are shown in red.

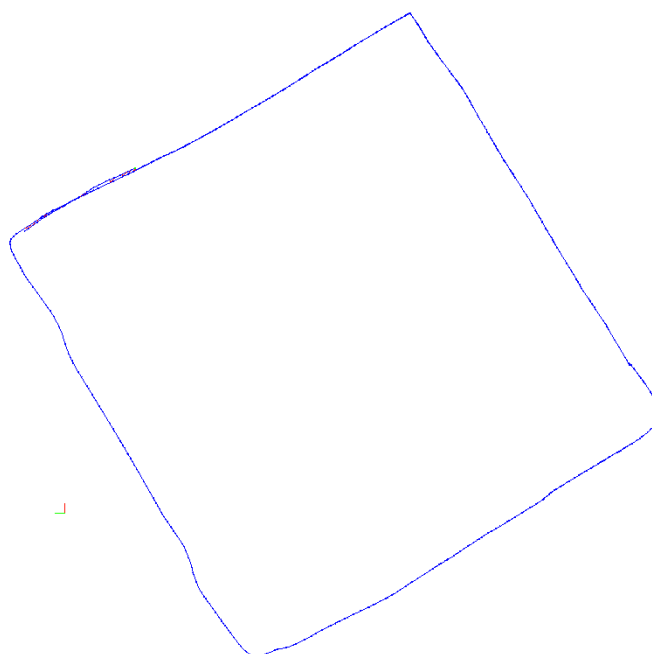


Fig. 6.14 Optimized loop. The corrected trajectory and link constraint can be seen.

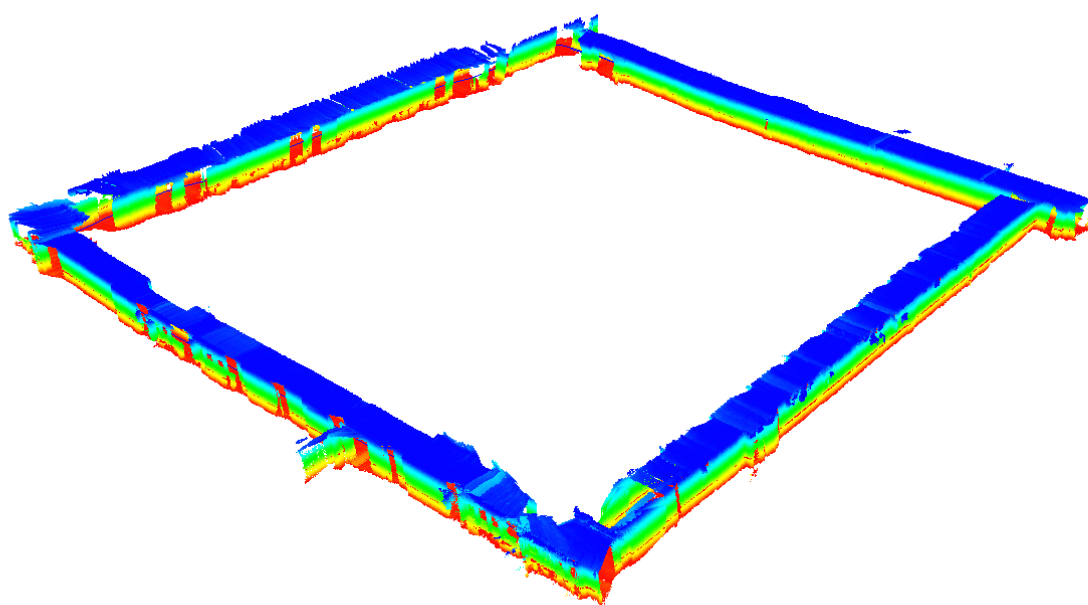


Fig. 6.15 3D map of assembled point cloud for a large loop.

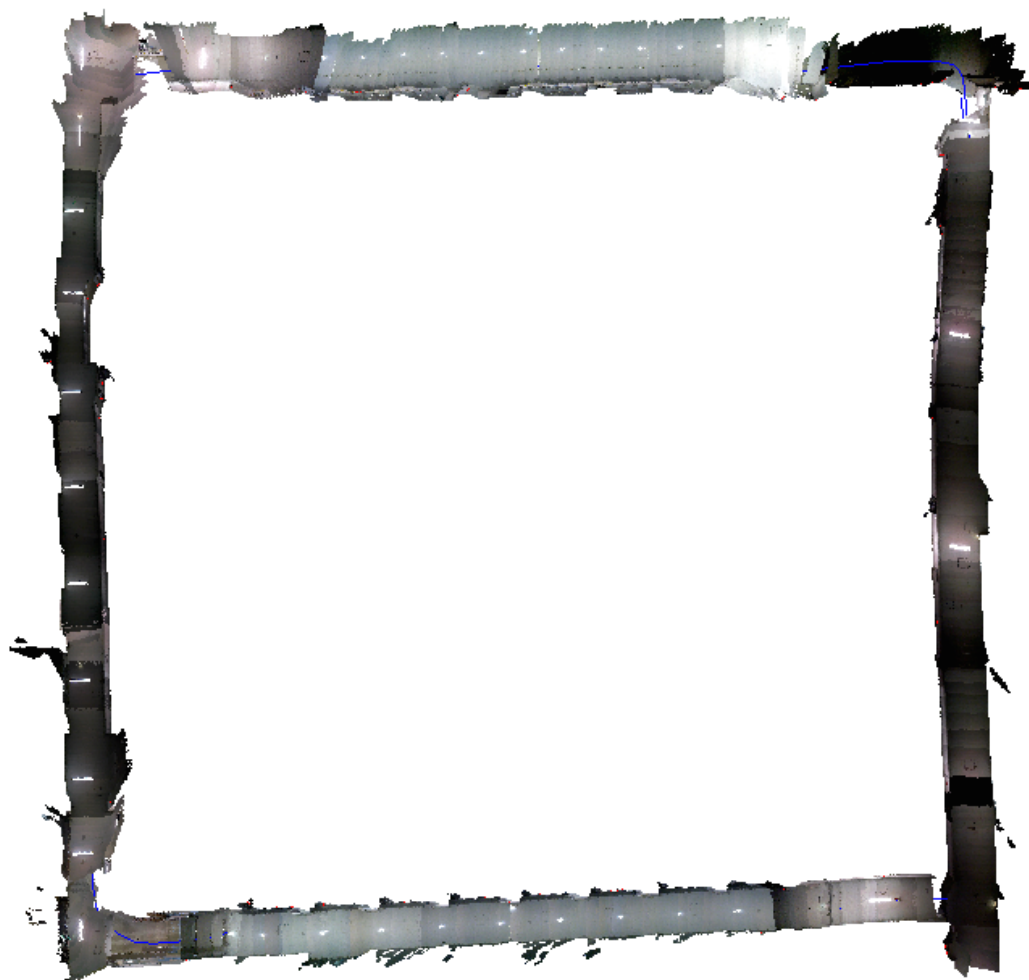


Fig. 6.16 Top view of the 3D map with colored point map.

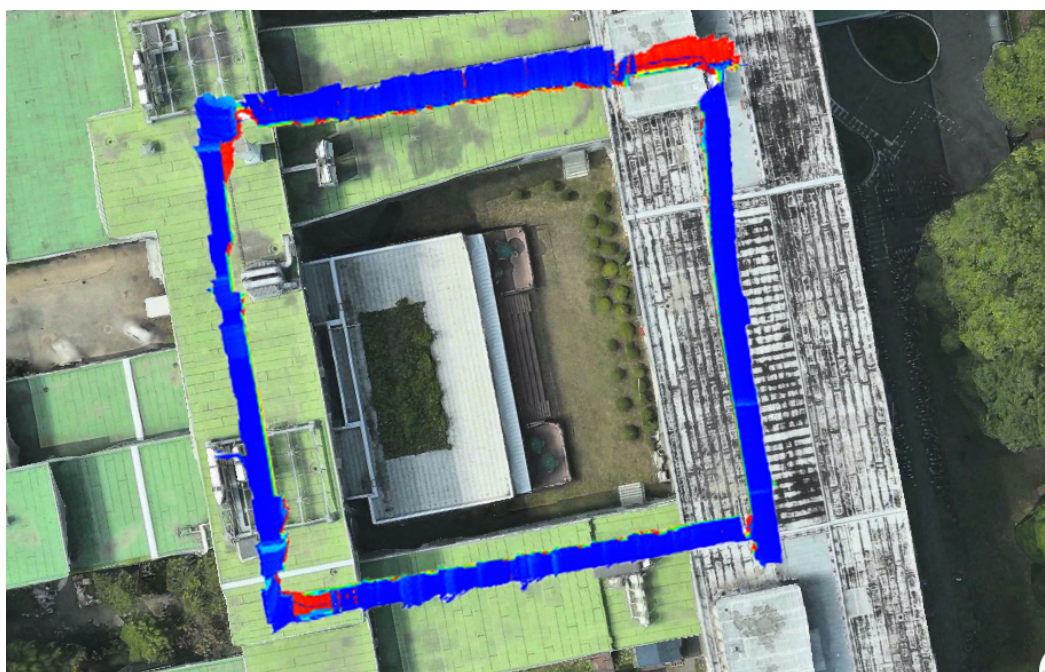


Fig. 6.17 Overlaid 3D map on actual building map, shows the accuracy of the map.

Chapter 7

Conclusion and Future Works

The work in this thesis discusses an important problem in mobile robotics called as the Simultaneous Localization and Mapping problem or the SLAM problem. SLAM is an interesting and large research field and this thesis work just scratches the surface of the amazing world of autonomous mobile robots. Although discreet, the work mainly covers the mapping problem in detail for a mobile robot working in a structured indoor environment. Probabilistic techniques and algorithms using the Extended Kalman Filter (EKF), Particle Filter, and graph theory are utilized to construct robust 2D maps of the indoor environments. The major work in this regard is the use of clustering techniques for noise removal from the laser scan data and for producing efficient line maps by clustering in the Hough domain. A standard line segment EKF based SLAM is used to construct the line segments using extracted features (corners and line segments) from the LRF data. In the end, a multi-sensor fusion technique is presented that uses a low cost depth vision sensor to construct 3D maps of the environment. The algorithm uses common vision techniques with scan matching from the LRF to achieve loop closure in case of large environment mapping. The algorithm works efficiently in a well lit indoor environments and is able to produce robust 3D maps that can be utilized for structural mapping of the indoor areas.

7.1 Future Works

- The present work does not include path-planning algorithms for the mobile robot. On top of the presented work, algorithms such as A^* or the D^* can be used for truly autonomous navigation. This can have great applications in service areas for delivering objects, exploration, search and security surveillance.

- The present work can only work in indoor environment due to the limitation of the robot to operate in a rugged areas and range of the laser sensor. Also, the depth sensors cannot operate in direct sunlight and its one of the bottlenecks of using depth sensor outside. The work can be extended to use a stereo vision system for truly outdoor experience.
- Presently there has been a great surge in vision based odometry SLAM solutions in the research community and this could be one of the potential future work.
- Object manipulation in the 3D environment is a challenging research and is extensively been studied by other researchers. Using the 3D world information from the map and by using manipulators on the robot, real objects can be manipulated for applications such as object detection, pick and place, recognizing doors and handles and operating simple tasks.
- The present system can be implemented to build a smart wheelchair for the service of the disabled. Using the autonomous navigation, people with severe motor disability can be benefited by such smart wheelchairs to work in an indoor environment.

References

- [1] Adams, M. and Kerstens, A. (1998). Tracking naturally occurring indoor features in 2-d and 3-d with lidar range/ amplitude data. *The International Journal of Robotics Research*, 17(9):907–923.
- [2] Anderson, B. and Moore, J. (1979). *Optimal filtering*. Prentice-Hall information and system sciences series. Prentice-Hall.
- [3] Arras, K., Castellanos, J., Schilt, M., and Siegwart, R. (2003). Feature-based multi-hypothesis localization and tracking using geometric constraints. *Robotics and Autonomous Systems*, (1056):1–13.
- [4] Arras, K. O. and Siegwart, R. Y. (1997). Feature extraction and scene interpretation for map-based navigation and map building. In *Proc. of SPIE, Mobile Robotics XII*, pages 42–53.
- [5] Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.
- [6] Bar-Shalom, Y. (1987). *Tracking and Data Association*. Academic Press Professional, Inc., San Diego, CA, USA.
- [7] Barkby, S., Williams, S. B., Pizarro, O., and Jakuba, M. V. (2011). A featureless approach to efficient bathymetric slam using distributed particle mapping. *Journal of Field Robotics*, 28(1):19–39.
- [8] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008a). Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359.
- [9] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008b). Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359.
- [10] Bezdek, J. C., Ehrlich, R., and Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2–3):191 – 203.
- [11] Bonin-Font, F., Ortiz, A., and Oliver, G. (2008). Visual navigation for mobile robots: A survey. *J. Intell. Robotics Syst.*, 53(3):263–296.
- [12] Borenstein, J., Everett, H. R., Feng, L., and Wehe, D. (1997). Mobile robot positioning sensors and techniques.
- [13] Borges, G. A. and Aldon, M.-J. (2004). Line extraction in 2d range images for mobile robotics. *J. Intell. Robotics Syst.*, 40:267–297.

- [14] Choi, C. and Christensen, H. (2013). Rgb-d object tracking: A particle filter approach on gpu. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1084–1091.
- [15] Clemente, L., Davison, A., Reid, I., Neira, J., and Tardós, J. (2007). Mapping Large Loops with a Single Hand-Held Camera. Atlanta, GA, USA.
- [16] Cox, I. J. (1991). Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *Robotics and Automation, IEEE Transactions on*, 7(2):193–204.
- [17] Crowley, J. L. (1995). Mathematical foundations of navigation and perception for an autonomous mobile robot. In *Reasoning with Uncertainty in Robotics*, pages 9–51. Springer-Verlag.
- [18] Cummins, M. and Newman, P. (2010). Invited Applications Paper FAB-MAP: Appearance-Based Place Recognition and Mapping using a Learned Visual Vocabulary Model. In *27th Intl Conf. on Machine Learning (ICML2010)*.
- [19] Davies, E. R. (2004). *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [20] Davison, A., Reid, I., Molton, N., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067.
- [21] Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-whyte, H. F., and Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17:229–241.
- [22] Douc, R. (2005). Comparison of resampling schemes for particle filtering. In *In 4th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 64–69.
- [23] Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15.
- [24] Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1 edition.
- [25] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110.
- [26] Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.
- [27] Eliazar, A. and Parr, R. (2003). Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI’03*, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

- [28] Ester, M., Peter Kriegel, H., S. J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press.
- [29] Fang, Z. and Zhang, Y. (2015). Experimental evaluation of rgb-d visual odometry methods. *Int J Adv Robot Syst*, 12:26.
- [30] Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.
- [31] Fernández, C., Moreno, V., Curto, B., and Vicente, J. A. (2010). Clustering and line detection in laser range measurements. *Robotics and Autonomous Systems*, 58(5):720 – 726.
- [32] Fischler, M. A. and Bolles, R. C. (1981a). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.
- [33] Fischler, M. A. and Bolles, R. C. (1981b). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.
- [34] Forsberg, J., Larsson, U., and Wernersson, A. (1995). Mobile robot navigation using the range-weighted hough transform. *Robotics Automation Magazine, IEEE*, 2(1):18–26.
- [35] Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [36] Fuentes-Pacheco, J., Ruiz-Ascencio, J., and Rendón-Mancha, J. (2015). Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81.
- [37] Gelb, A. (1974). *Applied Optimal Estimation*. M.I.T. Press.
- [38] Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113.
- [39] Grisetti, G., Kuemmerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43.
- [40] Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46.
- [41] Guo, S., Pridmore, T., Kong, Y., and Zhang, X. (2009). An improved hough transform voting scheme utilizing surround suppression. *Pattern Recogn. Lett.*, 30(13):1241–1252.

- [42] Han, J. (2005). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [43] Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2009). *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York. Autres impressions : 2011 (corr.), 2013 (7e corr.).
- [44] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663.
- [45] Högman, V. (2011). Building a 3d map from rgb-d sensors. Master’s thesis, Royal Institute of Technology, Sweden.
- [46] Hu, G., Huang, S., Zhao, L., Alempijevic, A., and Dissanayake, G. (2012). A robust rgb-d slam algorithm. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1714–1719. IEEE.
- [47] Huang, S. and Dissanayake, G. (2007). Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23(5):1036–1049.
- [48] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [49] Jang, J.-H. and Hong, K.-S. (2002). Fast line segment grouping method for finding globally more favorable line segments. *Pattern Recognition*, 35(10):2235–2247.
- [50] Jensfelt, P. and Jensfelt, P. (2001). Approaches to mobile robot localization in indoor environments. Technical report, PhD thesis, Signal, Sensors and Systems (S3), Royal Institute of Technology, SE-100 44.
- [51] Kailath, T., Sayed, A., and Hassibi, B. (2000). *Linear Estimation*. Prentice-Hall information and system sciences series. Prentice Hall.
- [52] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*.
- [53] Kerl, C., Sturm, J., and Cremers, D. (2013). Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. IEEE.
- [54] Kretzschmar, H., Grisetti, G., and Stachniss, C. (2010). Lifelong map learning for graph-based SLAM in static environments. *KI – Künstliche Intelligenz*, 24:199–206.
- [55] Kuemmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.
- [56] Laan, M., K. P. and Bryan, J. (2003). A new partitioning around medoids algorithms. *Journal of Statistical Computation and Simulation*, 73(8):575–584.

- [57] Lazanas, A. and Latombe, J.-C. (1995). Motion planning with uncertainty: a landmark approach. *Artificial Intelligence*, 76(1–2):287 – 317. Planning and Scheduling.
- [58] Leonard, J. J., Durrant-Whyte, H. F., and Pj, O. (1992). Directed sonar sensing for mobile robot navigation.
- [59] Lindeberg, T. (1994). *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, Norwell, MA, USA.
- [60] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110.
- [61] Lu, F. and Milios, E. (1997a). Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349.
- [62] Lu, F. and Milios, E. (1997b). Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18(3):249–275.
- [63] Lv, J., Kobayashi, Y., Emaru, T., and Ravankar, A. A. (2015). Indoor slope and edge detection by using two-dimensional ekf-slam with orthogonal assumption. *Int J Adv Robot Syst*, 12:44.
- [64] Lv, J., Kobayashi, Y., Ravankar, A. A., and Emaru, T. (2014). Straight line segments extraction and ekf-slam in indoor environment. *Journal of Automation and Control Engineering Vol*, 2(3).
- [65] Martinez, J. L., Gonzalez, J., Morales, J., Mandow, A., and Garcia-Cerezo, A. J. (2006). Mobile robot motion estimation by 2d scan matching with genetic and iterative closest point algorithms. *Journal of Field Robotics*, 23(1):21–34.
- [66] Mathworks (April, 2014). Eigenvalues and singular values. <http://www.mathworks.com/moler/eigs.pdf>.
- [67] Matthies, L. and Elfes, A. (1988). Integration of sonar and stereo range data using a grid-based representation. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 727–733. IEEE.
- [68] Maybeck, P. S., In, T., Form, A., Means, O. B., Mechanical, O., Photocopy, I., and S, M. P. (1979). Stochastics models, estimation, and control: Introduction.
- [69] Montemerlo, M. and Thrun, S. (2003). Simultaneous localization and mapping with unknown data association using fastslam. In *ICRA*, pages 1985–1991. IEEE.
- [70] NASA Jet Propulsion Laboratory, C. I. o. T. (2010). Mars lander.
- [71] Nguyen, V., Gächter, S., Martinelli, A., Tomatis, N., and Siegwart, R. (2007). A comparison of line extraction algorithms using 2d range data for indoor mobile robotics. *Auton. Robots*, 23(2):97–111.

- [72] Pathak, K., Birk, A., Poppinga, J., and Schwertfeger, S. (2007). 3d forward sensor modeling and application to occupancy grid based sensor fusion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2059–2064.
- [73] Pavlidis, T. and Horowitz, S. (1974). Segmentation of plane curves. *IEEE Transactions on Computers*, 23(8):860–870.
- [74] Pfister, S. T., Roumeliotis, S. I., and Burdick, J. W. (2003). Weighted line fitting algorithms for mobile robot map building and efficient data representation. In *In ICRA*, pages 14–19.
- [75] Pinto, M., Moreira, P. A., and Matos, A. (2013). *Robots Localisation in Indoor and Service Scenarios, Two Approaches: Landmark-Based and Three-Dimensional*. LAP Lambert Academic Publishing.
- [76] Premebida, C. and Nunes, U. (2005). Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications. *Robotica*, pages 17–25.
- [77] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5.
- [78] Ravankar, A., Hoshino, Y., Emaru, T., and Kobayashi, Y. (2011). Map building from laser range sensor information using mixed data clustering and singular value decomposition in noisy environment. In *System Integration (SII), 2011 IEEE/SICE International Symposium on*, pages 1232–1238.
- [79] Ravankar, A., Hoshino, Y., Emaru, T., and Kobayashi, Y. (2012). Robot mapping using k-means clustering of laser range sensor data. *Bulletin of Networking, Computing, Systems, and Software*, 1(1).
- [80] Ravankar, A., Kobayashi, Y., Lv, J., Emaru, T., and Hoshino, Y. (2014). An embarrassingly parallel hoppingwindow noise removing algorithm for lidar based robot mapping. In *SICE Annual Conference (SICE), 2014 Proceedings of the*, pages 307–312.
- [81] Ravankar, A., Kobayashi, Y., Ravankar, A., and Emaru, T. (2015a). A connected component labeling algorithm for sparse lidar data segmentation. In *Automation, Robotics and Applications (ICARA), 2015 6th International Conference on*, pages 437–442.
- [82] Ravankar, A. and Sedukhin, S. (2010). Mesh-of-tori: A novel interconnection network for frontal plane cellular processors. In *Networking and Computing (ICNC), 2010 First International Conference on*, pages 281–284.
- [83] Ravankar, A. A. (2012). Map building from laser range sensor information in noisy environment using mixed clustering, singular value decomposition and hough transform. Master’s thesis, Hokkaido University, Japan.

- [84] Ravankar, A. A., Hoshino, Y., Ravankar, A., Jixin, L., Emaru, T., and Kobayashi, Y. (2015b). Algorithms and a framework for indoor robot mapping in a noisy environment using clustering in spatial and hough domains. *International Journal of Advanced Robotic Systems, InTech*, 12(10.).
- [85] Ronse, C. and Serra, J. (2010). Algebraic foundations of morphology. In *Mathematical Morphology: From Theory to Applications*, pages 35–80. ISTE / J. Wiley & Sons.
- [86] Ruckert, E. A. (2009). Simultaneous localisation and mapping for mobile robots with recent sensor technologies. Master's thesis, Graz University of Technology, Graz, Austria.
- [87] Sack, D. and Burgard, W. (2004). A comparison of methods for line extraction from range data. In *In Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*.
- [88] Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-icp. In *Proceedings of Robotics: Science and Systems*, Seattle, USA.
- [89] Serra, J. (1983). *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA.
- [90] Shatkay, H. and Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. In *IN PROCEEDINGS OF IJCAI-97. IJCAI, INC*, pages 920–929.
- [91] Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA.
- [92] Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *Int. J. Rob. Res.*, 5(4):56–68.
- [93] Soille, P. (2003). *Morphological Image Analysis: Principles and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition.
- [94] Sorenson, H. W. and Stubberud, A. R. (1968). Non-linear filtering by approximation of the a posteriori density. *International Journal of Control*, 8(1):33–51.
- [95] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*.
- [96] Sünderhauf, N. and Protzel, P. (2012). Towards a robust back-end for pose graph slam. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1254–1261. IEEE.
- [97] Tan, P., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Pearson International Edition. Pearson Addison Wesley.
- [98] Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Auton. Robots*, 15(2):111–127.

-
- [99] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.
 - [100] Torr, P. H. S. and Zisserman, A. (2000). Mlesac: A new robust estimator with application to estimating image geometry. *Comput. Vis. Image Underst.*, 78(1):138–156.
 - [101] Walsh, D. and Raftery, A. E. (2002). Accurate and efficient curve detection in images: the importance sampling hough transform. *Pattern Recognition*, 35(7):1421–1431.
 - [102] Welch, G. and Bishop, G. (1995). An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA.
 - [103] Wikipedia (2010). Darpa grand challenge.
 - [104] Yu, J., Ku, W.-S., Sun, M.-T., and Lu, H. (2013). An rfid and particle filter-based indoor spatial query evaluation system. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 263–274, New York, NY, USA. ACM.
 - [105] Zhu, X. (2006). Semi-supervised learning literature survey.

Appendix A

Robot Odometry Model

A.1 Robot Odometry

The robot coordinate frame system is defined as shown in Figure A.1. The global and local coordinate system for the robot are represented by $O - xy$ and $O' - x'y'$, respectively. The diameter of the wheels is d and the tread is given by w . The midpoint of w is fixed at point O' , which is the origin of the local coordinate system $O' - x'y'$ fixed on the robot. The robot moves in the direction y' perpendicular to the direction x' . The angle θ is the angle between the y' axis and the x axis in a counter-clockwise direction. The rotational angle of the left wheel (ϕ_L) and the right wheel (ϕ_R) is defined in the forward direction of motion. The angular rotation of the robot is given by ω . The velocities of the wheels of the robot v_L and v_R are expressed as:

$$v_R = \dot{\phi}_R d, \quad v_L = \dot{\phi}_L d \quad (\text{A.1})$$

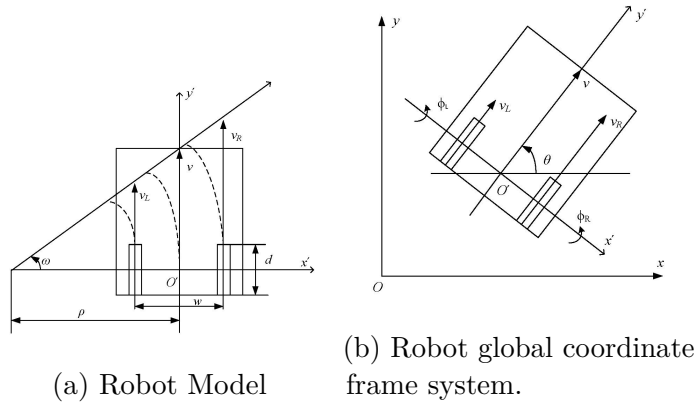


Fig. A.1 Robot coordinate frame system.

Here, $\dot{\phi}$ represents the differential operator with respect to time t . From Figure A.1a, considering the circular motion and the curvature radius ρ , we can define the velocity v as:

$$v = \rho\omega. \quad (\text{A.2})$$

On the other hand, using the tread w , v_L and v_R can be written as:

$$v_R = \left(\rho + \frac{w}{2}\right)\omega, \quad v_L = \left(\rho - \frac{w}{2}\right)\omega. \quad (\text{A.3})$$

Using Equations (A.1), (A.2) and (A.3), the following relationships are obtained:

$$\omega = \frac{(v_R - v_L)}{w}, \quad (\text{A.4})$$

$$v = \frac{(v_R + v_L)}{2}, \quad (\text{A.5})$$

and:

$$\rho = \frac{w(v_R + v_L)}{2(v_R - v_L)}. \quad (\text{A.6})$$

Here, assuming v and ω as constant values during time $[t_0, t_1]$, integrating Equations (A.5) and (A.6) in the interval $[t_0, t_1]$ yields the following equations:

$$(t_1 - t_0)\omega = \frac{(D_r(t_1) - D_r(t_0)) - (D_l(t_1) - D_l(t_0))}{w}, \quad (\text{A.7})$$

$$(t_1 - t_0)v = \frac{(D_r(t_1) - D_r(t_0)) + (D_l(t_1) - D_l(t_0))}{2}, \quad (\text{A.8})$$

where the displacement of the left and right wheels at time t is $D_l(t)$ and $D_r(t)$. From Equations (A.7) and (A.8), the robot's displacement and orientation with respect to $O - xy$ are given by:

$$\Delta\theta = \frac{(D_r(t_1) - D_r(t_0)) - (D_l(t_1) - D_l(t_0))}{w}, \quad (\text{A.9})$$

$$\Delta x = \frac{(D_r(t_1) - D_r(t_0)) + (D_l(t_1) - D_l(t_0))}{2} \cos \theta(t_0), \quad (\text{A.10})$$

$$\Delta y = \frac{(D_r(t_1) - D_r(t_0)) + (D_l(t_1) - D_l(t_0))}{2} \sin \theta(t_0). \quad (\text{A.11})$$

Assuming that $\Delta\theta$ is sufficiently small, from Equations (A.9), (A.10) and (A.11), we get the displacement from time $[t_0, t_1]$. The sensor data at time t_0 is represented at $O - xy$ and the sensor data at time t_1 is represented by $O' - x'y'$. In the next section, these parameters $(\Delta x, \Delta y, \Delta\theta)$ will be used to define algorithms for map generation.