View-based mobile robot navigation : a genetic algorithmic approach

Dong, Jun Feng

2011

Dong, J. F. (2011). View-based mobile robot navigation : a genetic algorithmic approach.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/46711>

<https://doi.org/10.32657/10356/46711>

**VIEW-BASED MOBILE ROBOT NAVIGATION:
A GENETIC ALGORITHMIC APPROACH**

DONG JUN FENG

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2011

ACKNOWLEDGEMENT

First and foremost I offer my sincerest gratitude to my supervisor, Prof Wijerupage Sardha Wijesoma, who has supported me throughout my research and writing up of my thesis with patience and guidance. Sorry for my poor descriptions, one just simply could not wish for a friendlier supervisor. As the Chinese sayings, 'one day teacher, whole life father', is what I feel today. I wish him good health and happiness.

Dr Andrew Shacklock, my co-supervisor, gave me useful guidance during my research.

In my research work, I have been blessed with a friendly and cheerful group of colleagues, lab technicians and fellow students who support me in various aspects. Mr Tan Chia Soon, a nice technician who helped me in many research projects is the first one that I must thank. His kind help in the last five years has been a great support for my research work. Dr Lee Kwang Wee provided me the testing data from 2D laser range finders and sonar sensors. I would also like to thank Mr. Bharath Kalyan as well. He has helped me with useful tips with linux software and hardware administration. Peyman Moghadam worked with me in vision navigation and sensor perception. He solved many problems encountered with vision which directly helped my research.

Besides, I want to thank the Fantastic Four team members in the TechX challenge. Without their efforts, there would not have been an integrated testing platform, X-warrior packbot, for some of the experiments.

The Department of Control and Instrumentation, School of Electronics and Electrical Engineering (EEE), has provided the support and equipment I have needed to produce and complete my thesis.

Finally, I thank my parents and my girl friend for encouraging me throughout all my studies at University, especially when I was writing this thesis.

TABLE OF CONTENTS

<i>Acknowledgement</i>	<i>ii</i>
<i>Table of Contents</i>	<i>iii</i>
<i>Summary</i>	<i>v</i>
<i>List of Figures</i>	<i>vii</i>
<i>List of Tables</i>	<i>x</i>
1. Chapter I: Introduction	1
1.1. Mobile Robot Navigation	1
1.2. Rao-Blackwellized Particle filter SLAM Approaches	2
1.3. Non-feature based mapping for view based SLAM.....	3
1.4. Parallel Computing	5
1.5. Our Approach	5
1.6. Overview of thesis.....	7
1.7. Contributions of the thesis	7
2. Chapter II: Background	9
2.1. Motivation	9
2.2. Basic SLAM process.....	9
2.3. SLAM Approaches	11
2.4. Map representation and environmental mapping	14
2.5. Rao-Blackwellized particle filters SLAM.....	25
2.6. Summary	39
3. Chapter III: Genetic Algorithmic Filter Mapping	40
3.1. Motivation	40
3.2. Compressed particle filter mapping	40
3.3. Genetic algorithmic filter mapping.....	44
3.4. Performance comparisons in 2D	61
3.5. 3D GAF mapping results.....	76
3.6. Summary	84
4. Chapter IV: Rao-Blackwellized Genetic Algorithmic Filter SLAM	86
4.1. Introduction.....	86
4.2. Basic formulations.....	87
4.3. Description of the RBGAF-SLAM.....	90
4.4. Computational complexity	100
4.5. Illustration of RBGAF-SLAM by a simple example	102

4.6.	Simulation and experimental results	112
4.7.	Improvements for RGBAF-SLAM	136
4.8.	Summary	146
5.	<i>Chapter V: High Speed RGBAF-SLAM</i>	148
5.1.	Motivation	148
5.2.	Background knowledge about CUDA.....	150
5.3.	High Speed RGBAF-SLAM.....	157
5.4.	Experimental results	163
5.5.	Summary	165
6.	<i>Chapter V: Discussion and Conclusions</i>	166
6.1.	Discussion	166
6.2.	Conclusions.....	168
6.3.	Suggestions for future research	170
	<i>Publications</i>	171
	<i>References</i>	172
	<i>Appendix A: Velodyne 3D LIDAR Processing Algorithms</i>	179
	<i>Appendix B: Sample CUDA SIMD Program</i>	190

SUMMARY

This thesis examines the simultaneous localization and mapping (SLAM) problem. This is the task of incrementally mapping an unknown environment whilst simultaneously localizing the mobile robot with respect to this map.

Currently, much research is focused on feature (landmark) based SLAM approaches. These approaches, like Extended Kalman filter (EKF) SLAM and FastSLAM, are well investigated and implemented by many researchers, proving to be effective in some environments. However, feature based approaches are limited as to how successful the environmental feature extraction process is. To extract reliable features, some prior knowledge is assumed about the environmental features. Further, in a highly complex and unstructured environment, feature extraction approaches can only make use of a very small portion of the measurement, without fully exploiting the rich information provided by the sensor. Non-feature based methods are also known as view based SLAM approaches, and can either complement or replace feature based SLAM.

Although view based SLAM does not require the feature extraction process, an effective mapping approach is needed to map the unknown environment. Occupancy grid mapping is the common mapping approach exploited in non-feature based SLAM, though it is computationally expensive. An alternative to using a grid for map representation is a particle map in non-feature based SLAM. It is more efficient than occupancy grids, especially in mapping 3D environments. However, the particle map representation suffers from the sample impoverishment problem in its re-sampling process and this problem is difficult to solve by analytic or simple heuristic means. In our research, the mapping process is viewed as a complex non-linear optimization problem. Inspired by the success of Genetic Algorithms (GA) for solving complex and highly non-linear optimization problems in other application areas, we use GA to solve the mapping problem so in the proposed view based SLAM.

In the context of GA, the problem of solving sample impoverishment in a particle map is equivalent to solving the local maxima problem through global optimization. The advantage of applying GA is that it has been highly developed with many sophisticated techniques which can balance the divergence and convergence of the sample sets and avoid local maxima in the global search. These techniques from GA are applied in a systematic manner to mitigate the sample impoverishment problem of particle maps, resulting in our development of an efficient Genetic Algorithmic Filter (GAF) mapping for view based SLAM approaches. The strategy and the resulting GAF is very effective and efficient, and especially appropriate for 3D mapping.

This GAF approach to mapping is then used to develop an efficient real-time view-based SLAM approach known as the Rao-Blackwellized Genetic Algorithmic Filter (RBGAF) SLAM. The basic concept is to combine a Rao-Blackwellized particle filter with GAF mapping for SLAM. The Rao-Blackwellized particle filter provides the theoretical foundation of the SLAM approach used with GAF. The GAF provides higher efficiency in mapping compared to occupancy grid, especially for 3D SLAM.

Due to the requirement of high computational resources, a parallel computing architecture is developed for RBGAF-SLAM on a graphical processing unit (GPU), achieving real-time performance on a standard PC equipped with a GPU. The implementation is computationally efficient and effective even in highly complex and uncertain environments. A series of simulations and experimental results with various types of sensors and scenarios is presented to demonstrate the effectiveness and efficiency of the proposed approach in different environments.

LIST OF FIGURES

Figure 2-1: Graphical model of concurrent mapping and localization.....	11
Figure 2-2: Uncertainty of point feature.....	16
Figure 2-3: Range and bearing measurement for feature L in 3D	16
Figure 2-4: Feature based SLAM in curve-like environment.....	19
Figure 2-5: Under water sonar image.....	20
Figure 2-6: Isohypse lines for terrain representation, heights in meters.....	21
Figure 2-7: An example of an occupancy grid map.	24
Figure 3-1: Illustration of a genetic algorithm	45
Figure 3-2: Illustration of a genetic algorithm	47
Figure 3-3: 2D Cartesian coordinates.....	48
Figure 3-4: 2D Cartesian coordinates.....	49
Figure 3-5: Illustration of 1D range sensor model.....	54
Figure 3-6: Example of 2D Player/Stage simulation environment.	62
Figure 3-7: SICK 2D laser range finder	64
Figure 3-8: Simple 1D sensor model.....	65
Figure 3-9: Normal distribution $N(0,2)$	66
Figure 3-10: Biased distribution for the errors $N(2,2)$	67
Figure 3-11: Line feature extraction in simulation.....	68
Figure 3-12: Occupancy grid mapping result for a static robot.	69
Figure 3-13: Occupancy grid map.....	70
Figure 3-14: Number of particles for a particle map.....	71
Figure 3-15: Monte Carlo resampling results.	72
Figure 3-16: Histogram for the count values of compressed particle map.....	72
Figure 3-17: Simulation results of GAF mapping.....	74
Figure 3-18: The number of chromosomes for single GAF map.....	75
Figure 3-19: Number of map chromosomes in GAF vs. number of grid cells	76
Figure 3-20: Velodyne 3D laser range finder	77
Figure 3-21: NTU campus map: experiment environment for Velodyne 3D LiDAR.	78
Figure 3-22: Number of map chromosomes for static 3D mapping.....	79
Figure 3-23: 3D GAF mapping result with location A and B marked (45 degree view)	80
Figure 3-24: 3D environment in campus view with location A marked	80
Figure 3-25: 3D environment in campus view with location B marked	81
Figure 3-26: Details of 3D GAF mapping results. (45 degree view)	81
Figure 3-27: Number of chromosomes for various virtual chromosome sizes.....	82
Figure 3-28: 3D mapping result with virtual chromosome size 60 cm (45 degree view)	83
Figure 3-29: 3D mapping result with virtual chromosome size 10 cm (45 degree view)	83
Figure 3-30: Relative computational cost (the cost at size 60 cm is set at 100).....	84
Figure 4-1: Illustrated RBGAF-SLAM procedures.....	93
Figure 4-2: A two-wheel robot with width L and wheel radius r.....	94
Figure 4-3: Simple motion model for two-wheel mobile robot.....	95
Figure 4-4: Simple example of a robot moving towards a wall.....	103

Figure 4-5: Rao-Blackwellised particles at time $t=0$	104
Figure 4-6: Three robot particles estimated at time $t=1$	105
Figure 4-7: Map update for Rao-Blackwellised particle $x_1^{(0)}, m_1^{(0)}$ at time $t=1$	108
Figure 4-8: Map update results for Rao-Blackwellised particles.....	109
Figure 4-9: tournament selection between Rao-Blackwellised particles $x^{(0)}$ and $x^{(2)}$	111
Figure 4-10: Three Rao-Blackwellised particles after RBGAF update at time $t=1$	112
Figure 4-11: Simulation environment for SLAM	113
Figure 4-12: RBGAF-SLAM result.....	114
Figure 4-13: Vertical axis: localization error in x (m) vs. horizontal axis: time (s).....	114
Figure 4-14: Vertical axis: localization error in y (m) vs. horizontal axis: time (s).....	115
Figure 4-15: Segway two-wheel robot	116
Figure 4-16: Experimental environment at NTU-S2.....	116
Figure 4-17: Mapped environment.....	117
Figure 4-18: Packbot robot	118
Figure 4-19: RBGAF-SLAM result.....	118
Figure 4-20: SLAM result compared with real environment.....	119
Figure 4-21: Underwater sonar image	120
Figure 4-22: RBGAF-SLAM result with sonar sensors	121
Figure 4-23: Localization errors with sonar sensors	122
Figure 4-24: Ground truth and mapping result (45 degree view)	128
Figure 4-25: Mapping result with large heading noise (45 degree view).....	128
Figure 4-26: Trajectories filtered by latest two laser scans (45 degree view)	129
Figure 4-27: RBGAF-SLAM result with very large heading noise (45 degree view)	130
Figure 4-28: Heading errors of RBGAF-SLAM test one	130
Figure 4-29: RBGAF-SLAM result (in NTU campus, top view)	131
Figure 4-30: RBGAF-SLAM test result (45 degree view)	132
Figure 4-31: Trajectory from the odometry readings used in SLAM	132
Figure 4-32: Localization error in x axis	133
Figure 4-33: Localization error in y axis	133
Figure 4-34: RBGAF-SLAM result with 2D obstacle map (top view)	136
Figure 4-35: Experimental result of pre-processing and sensor rejection	139
Figure 4-36: Influence of selection method and resampling frequency	141
Figure 4-37: Genetic learning algorithm for RBGAF-SLAM	142
Figure 4-38: Mean loop-closing error through genetic learning.....	146
Figure 5-1: Processing speed of GPU vs. CPU	151
Figure 5-2: Performance / price ratio for CPU & GPU	152
Figure 5-3: Logical architecture of CUDA kernels	155
Figure 5-4: Natural data structure for RB particle (trajectory and map pair)	158
Figure 5-5: CUDA kernels for map selection	161
Figure 5-6: CUDA kernels for map chromosome generation.....	162
Figure 5-7: CUDA kernels for RB-Particle evaluation	162

Figure 5-8: CUDA kernels for final resampling	163
Figure 7-1: Overall Processing System Block Diagram.....	181
Figure 7-2: The overall block diagram of the obstacle/ground classification module.	183
Figure 7-3: GUI view of obstacle detection result using 3D Velodyne data.....	184
Figure 7-4: Surface/Slope Estimation.....	185
Figure 7-5: GUI view of surface/slope detection results.	186
Figure 7-6: GUI view of curb line (white) extracted.....	189
Figure 8-1: CUDA architecture with block and thread.	190

LIST OF TABLES

Table 3-1: Major variable notations for GAF mapping.....	61
Table 3-2: Compare mapping result with ground truth.....	84
Table 4-1: Variable notations.....	92
Table 4-2: Comparison between SLAM algorithms	101
Table 4-3: SLAM results with various virtual chromosome sizes.....	134
Table 4-4: Localization errors with virtual measurement noises.....	134
Table 4-5: Genetic code for optimizing RBGAF-SLAM	144
Table 5-1: Computational power for CPU & GPU	152
Table 5-2: Processing time of SLAM approaches.....	164

CHAPTER I: INTRODUCTION

1.1. Mobile Robot Navigation

Robotics is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices. Mobile robots which navigate and explore the environment are one type of robotic system. In particular, mobile robots have great applications in industry, transportation and aerospace exploration. To complete the assigned tasks, there is one fundamental problem for mobile robots to be solved - ‘where am I?’ This is often known as the localization problem. There are basically two categories of models for this problem: metric and topological. In the topological model, the environment is represented by a finite number of ‘nodes’ and ‘links’. In the metric model, the environment is represented as a two-dimensional or three-dimensional space, usually in Cartesian coordinates. Our work focuses on the metric model and in this thesis all the discussion will be based on an environmental space represented by Cartesian coordinates.

There are various localization methods for different scenarios. The direct solution is to estimate the robot location based on odometry information. However, the localization error from this solution is large and unbounded. With man-made active beacons or artificial landmarks, the localization problem can be greatly simplified as with the Global Positioning System (GPS). If the environmental map can be obtained before navigation starts, the localization problem can be simplified into map-aided localization or map-based localization. There are extensive researches in this area and great success has been achieved in many applications. However, localization in an unknown environment is still difficult. In such a case, the robot has to simultaneously localize itself and map the environment for future navigation. This is called simultaneous localization and mapping (SLAM) [1] or concurrent localization and mapping.

Simultaneous localization and mapping (SLAM) is the process by which a mobile robot builds an environmental map while simultaneously determining its pose

(position and orientation) within this map. SLAM is also known as Concurrent mapping and localization (CML) [2]. The success of SLAM is important for the success of mobile robot navigation in unknown environments [3]. The reasons are based on two well known facts. Firstly, the localization accuracy of odometry [4] or inertial navigation systems (INS) [5] is not enough for most mobile robots to complete their tasks. Localization errors will be accumulated and be unbounded. Secondly, an accurate environmental map is not available in an unknown environment. Although global positioning systems (GPS) [6] can provide position information, their accuracy and availability is not good enough [7]. For example, it does not work in an indoor environment. Therefore, research into SLAM is important. In this thesis, our research focuses on the SLAM problem for an efficient, effective and robust SLAM solution for mobile robot navigation.

1.2. Rao-Blackwellized Particle filter SLAM Approaches

The practical objective of simultaneous localization and mapping (SLAM) algorithms is to estimate the joint posterior of robot pose and environmental map consistently and efficiently. Murphy, Doucette and colleagues [8] introduced the *Rao-Blackwellized* particle filter (RBPF). This performs a factorization of the SLAM problem. The Rao-Blackwellized particle filter approach has been proven to be an effective means for estimating the full SLAM posterior and has since been extended and implemented in robot SLAM by many researchers.

Montemerlo and colleagues [1] proposed a Rao-Blackwellized particle filter-based SLAM approach known as FastSLAM. Here particles are used to represent the posterior over robot paths. The conditional independence of features (landmarks) given the robot pose is exploited to factor the mapping problem into separate problems, one for each feature in the map. The map features are estimated using Extended Kalman Filters (EKF). This well-known algorithm has since been proven to be a very efficient means of solving the SLAM problem.

However, FastSLAM requires reliable feature extraction. This requires some prior knowledge of the environment and will be very unreliable in a highly complex unknown environment. This drawback was later overcome by Hahnel [9]. Hahnel

used raw laser scans, instead of features, in FastSLAM to avoid feature extraction. Consecutive scans were matched to find the change in robot pose. A parametric model was used to determine experimentally the uncertainty associated with scan matching, and hence the uncertainty in the predicted robot poses. To yield consistent results, it is imperative that the pose estimation errors from matching scans are independent of the number of observations and nature of the environment, which are not really premises of the probabilistic approach to robotics. The method may work provided the experimental environment and actual environment are very much alike and the number of observations is about the same during the actual navigation.

The feature-based, sometimes called landmark-based, SLAM approaches require reliable features extracted from unknown environment; this requirement relies on a weak assumption that people can obtain prior knowledge for the success of feature extraction from unknown environment. Scan matching [10], on the other hand, provides a solution for performing SLAM with raw exteroceptive sensor measurements. However, to model highly complicated, non-Gaussian, non-linear sensor measurement is difficult and complicated in the scan-matching framework. Non-feature-based, also known as view-based SLAM does not require the feature extraction process, but it relies on an effective non-parametric approach to represent and map the unknown environment. Therefore, it is important to investigate non-parametric mapping approaches for view based SLAM with raw sensor measurements and, at the same time, to be able to implement sophisticated sensor measurement models.

1.3. Non-feature based mapping for view based SLAM

Occupancy grids [11] provide a way of solving the SLAM problem without feature extraction while preserving the potential to model highly complicated sensor measurements. It is the most popular non-parametric mapping method implemented in view based SLAM. Grisetti [12] proposed adaptive versions and selective re-sampling techniques to reduce the number of particles with occupancy grid mapping. However, the advantages of particle reduction are somewhat outweighed

by the method's requirement for high-resolution grids to represent the maps. Another Rao-Blackwellized approach to SLAM using occupancy grid maps is due to Eliazar [13] and known as DP-SLAM. It requires only one single grid map. However, the large computational cost required by occupancy grid map-based approaches restricts its usage, especially in 3D environments. Robot navigation often takes place in a large environment and requires high-resolution map representations. The situation would become even worse if implemented in a 3D environment.

Besides occupancy grid mapping, particle filters, also a non-parametric method, are frequently used to estimate the state of a dynamic system and is used for environmental mapping as well. The advantages of such particle mapping in SLAM are three-fold. Firstly, they are more efficient at approximating the map posterior. The map re-sampling process from particle filters will reduce the number of particles needed, resulting in a more efficient map representation. Secondly, in a particle map representation, the computational cost increases with the number of objects observed but not the size of the space navigated. The approach strips out 'unimportant' areas like 'empty space' and 'unknown space' from the environmental map so that both computational time and memory consumption can be reduced with negligible penalty in positioning accuracy. The third important advantage is that, unlike occupancy grid cells, map particles can be stored in any sequence and can be easily separated, combined, interchanged or deleted. This advantage greatly eases the hardware implementation of a particle map. Therefore, the particle map can be processed by parallel computing efficiently with simple data structures.

However, the particle map is not as popular as the occupancy grid map due to drawbacks such as the sample impoverishment problem. Due to the limitations of particle re-sampling, it is necessary to design adaptive techniques to reduce the real-time computational costs without loss of performance. Apart from sample size, sample impoverishment is a major issue with particle filter-based approaches. Heuristic methods, like sample boosting, have been introduced, as in [14], to mitigate these problems. However, the environmental map could be very complex

and it varies very much in space size. During all the resampling processes, map representation requires the particle set to maintain ‘suitable’ divergence which is quite difficult for these heuristic methods. Therefore, very few researchers apply particle filters to map representation in SLAM. Usually, particle filters are applied to represent robot posterior. This is because it is easier to achieve ‘suitable’ divergence to represent robot poses. If more effective techniques can mitigate these problems, particle filters will be applied to map representation that can improve SLAM performance.

1.4. Parallel Computing

Parallel computing is an effective way of solving a task requiring large computational resources. The current Graphical Processing Units (GPUs) provide very large computing power. However, such computational resources cannot be fully utilized if the algorithm is not properly designed to match the GPU computational architecture. Unlike the occupancy grid map, a particle map makes GPU processing much easier using off-the-shelf GPUs.

1.5. Our Approach

Although the feature based map is efficient and effective in certain environments with prior environment knowledge, mobile robot navigation in complex, unstructured, unknown environments requires view-based mapping approaches. Besides occupancy grid mapping, particle filters can be applied to mapping the environment. It has benefits and advantages over occupancy grids. However, the difficulties of variable particle sample size and the sample impoverishment problem restrict their use.

Our approach to overcome these deficiencies is to view the mapping process as a complex non-linear optimization problem. Genetic algorithms (GA) have proved to be effective in this regard. Genetic algorithms mimic the Darwinian selection of the fittest found in nature, using natural selection as paradigm for solving global optimization problems. It is guaranteed to converge to a good approximation of the solution, and it can be applied to any search space. The advantage of applying GA

is that it has been highly developed with many sophisticated techniques which can balance the divergence and convergence of the sample sets and avoid local maxima in the global search. GA techniques have been proven by applications in other research areas. In our approach, these techniques from Genetic Algorithms have been applied to mitigate problems with compressed particle filters, resulting in an efficient genetic algorithmic filter (GAF). We combine the genetic algorithmic filter (GAF) approach with compressed particle filter techniques for view-based map representation as opposed to occupancy grid mapping. The strategy and the resulting algorithm is very effective and efficient, and especially appropriate for 3D mapping. It provides a much more suitable way of representing a non-parametric environmental map for navigation.

When Rao-Blackwellized Particle filter SLAM is combined with a Genetic Algorithmic Filter for map approximation, we get the proposed Rao-Blackwellized Genetic Algorithmic Filter SLAM (RBGAF-SLAM). This approach benefits from both GAF mapping and the RBPF framework. In a view based SLAM algorithm, raw sensor measurements without any feature extraction process is required. Compared to a similar view based approach such as the occupancy grid map based SLAM approach DP-SLAM [13], the proposed technique is much more efficient in memory consumption and especially appropriate for 3D space. That is, with the same computational resources, more Rao-Blackwellized particles can be processed to provide more accurate SLAM results faster.

We adopt a Graphical Processing Unit (GPU) which has a highly parallel architecture very suitable for the proposed RBGAF-SLAM algorithm. To harness the advantages of RBGAF-SLAM and achieve real-time performance on a space- and weight-limited platform, we propose a highly parallel processing paradigm based on a graphics engine running on GPUs known as CUDA (*Compute Unified Device Architecture*). This results in a significant boost in speed and guarantees real-time processing. RBGAF-SLAM has been implemented in different environments with various types of sensors and provides very good performance.

1.6. Overview of thesis

Chapter II gives a brief literature review and outlines background knowledge related to Rao-Blackwellized particle filter SLAM (RBPF-SLAM). It discusses mobile robot SLAM in general and various common mapping approaches in SLAM. RBPF-SLAM approaches including feature based Fast-SLAM and view based DP-SLAM are described and discussed in detail. The pros and cons of each approach are also analysed and discussed.

Chapter III proposes our efficient Genetic Algorithmic Filter mapping based on Genetic Algorithms (GA). The effectiveness and the efficiency of this GAF mapping are shown by comparing various mapping approaches with simulations and experiments.

Chapter IV presents the proposed Rao-Blackwellized Genetic Algorithmic Filter for SLAM, based on the GAF mapping presented in Chapter III and is followed by its computational complexity analysis. Some 2D simulation results, 2D experimental results and 3D experimental results are presented to demonstrate that RBGAF-SLAM provides outstanding performance in various environments with various types of sensor. This chapter demonstrates that RBGAF-SLAM is an efficient means of solving the SLAM problem with a non-parametric map representation, especially appropriate in 3D SLAM.

In Chapter V, RBGAF-SLAM and its adaptation for highly parallel implementation on a Graphical Processing Unit (GPU) for real-time execution is described. The results shows that it provides a significant speed boost for RBGAF-SLAM.

Chapter VI presents the conclusions of the thesis.

1.7. Contributions of the thesis

The specific research contributions of this thesis to the problem of *simultaneous localization and mapping* (SLAM) may be summarized as follows:

1. Development of an efficient and flexible genetic algorithmic map representation for view-based SLAM approaches. It avoids the sample impoverishment problem of Monte Carlo resampling and is more efficient than occupancy grid maps, which is especially appropriate in a 3D environment.
2. Synthesis of an efficient real-time view-based SLAM approach: the Rao-Blackwellized Genetic Algorithmic Filter SLAM. Such a SLAM approach does not rely on features and it is capable of integrating arbitrary sensor and motion models.
3. Development of a highly efficient parallel computing structure for real-time RBGAF-SLAM. The approach can be implemented on a GPU which has much more computational resources and this provides a significant improvement in processing speed SLAM. As a result, real time 3D view based SLAM can be achieved.

CHAPTER II: BACKGROUND

2.1. *Motivation*

We focus on the metric based mobile robot navigation. In the past two decades, there are various metric based simultaneous localization and mapping (SLAM) approaches developed. The background knowledge as well as the state of the art on SLAM are introduced in this chapter. More specifically, this chapter focuses on the latest Rao-Blackwellized Particle Filter (RBPF) based SLAM approaches and introduced them in detail.

2.2. *Basic SLAM process*

2.2.1. *Metric vs. topological framework*

Classically, there are two categories for modeling the localization problem for mobile robots: metric and topological.

There are many techniques that have been proposed for topological maps. In [15], a general approach to indoor mobile robot navigation that facilitates navigation in unmodified environments of several hundred square meters, using simple sonar and odometry sensors is proposed. During the map-building phase, the user guides the robot around the environment and the system constructs the topological map. A three layer planning navigation method for indoor mobile robots is presented in [16]. On the other hand, in [17], visual images are used for topological navigation of urban vehicles which drives in outdoor environment. Autonomous navigation is quite successful with learning process.

A global topological map of the known environment is built to provide global navigation information for the robot. In recent years, the method of integrating topological and metric maps for mobile robot navigation has been widely used. According to the practical result of geometric-topological map building, a hybrid localization approach is presented to complete both extended Kalman-filter based

pose tracking in a local geometric map and re-localization in topological mode using the binding of high-level features [18]. This method is effective for avoiding localization failure in the case of unexpected collisions or in unstructured dynamic environments.

Though topological maps have many advantages, research in the metric framework is also important for autonomous mobile navigation. In the metric framework, the environment is represented as a set of objects with coordinates in a two-dimensional or three-dimensional space, usually in Cartesian coordinates.

In two dimensional space, the robot is assumed to move in a flat plane where the elevation information is ignored for all the environmental objects as well as the robot. The bearing angle of the robot platform is taken into consideration whereas the roll and pitch are usually ignored. In a three dimensional model, the elevation information is also taken into consideration with the roll pitch yaw values. Due to the precise representation of the metric framework, the present thesis focuses on the metric representation.

2.2.2. *Metric based SLAM*

Since the 1990s, probabilistic approaches derived from the recursive Bayes rule have become dominant in SLAM. The reason is that both robot mapping and localization are affected by sensor uncertainties [19], either due to sensor characteristics or to environmental noise. Probabilistic approaches can take these uncertainties into consideration by modeling sensor noise and its effects on the measurement. The progress of SLAM in the last two decades shows the advantages of probabilistic approaches. Hence this thesis also discusses probabilistic SLAM approaches. In probabilistic approaches, the SLAM process requires the robot control u , the sensor observation z about the environment. Figure 2-1 illustrates the basic procedures of SLAM.

Figure 2-1: Graphical model of concurrent mapping and localization

In the initial condition, the environmental map is unknown. The SLAM process starts with an initial robot pose x_0 which is a vector representing the robot pose at the discrete time instance $t=0$. If the initial robot pose to the global coordinate is unknown, the initial robot pose is usually assumed to be the origin of the global coordinates. With the robot control u , the new robot pose is estimated with some motion model (proposal distribution) and filtered by the SLAM algorithm with a measurement model to simultaneously determine the new robot pose and the updated environmental map.

2.3. SLAM Approaches

As presented in [3], there are three categories of SLAM approaches: Kalman Filters (KF), Expectation Maximization (EM) and Particle Filters (PF).

2.3.1. Kalman Filter based SLAM approaches

Kalman filters [20] are Bayes filters that represent posteriors using Gaussians. If Kalman filters are applied to solve SLAM problems, the assumption is that the state transition and the measurement functions are linear with Gaussian noise. In addition, the initial posteriors are also Gaussian. It is well known that there are two major variants of KF in SLAM approaches: the Extended Kalman Filter (EKF) [21] and the (Extended) Information Filters (IF or EIF) [20].

The EKF approximates the robot motion model using linear functions and accommodates the nonlinearities from the real world. It is quite popular in robotics

SLAM. Compared to EKF SLAM, IF filters are more accurate and more stable. However, IF has to calculate the inversion of the information matrix which is quite computationally expensive. This is the primary disadvantage of IF.

For SLAM, there are a number of variants of EKF approaches. For example, the unscented Kalman filter (UKF) [22] addresses the approximation issues of the EKF and the linearity assumption of the KF. In the UKF the state distribution is represented by a Gaussian Random Variable (GRV), but specified using a minimal set of carefully chosen sample points. On the other hand, the Compressed Extended Kalman Filter (CEKF) algorithm significantly reduces the computational requirement without introducing any penalties in the accuracy of the results.

In KF based SLAM approaches, features, often known as landmarks, are extracted from sensor measurements and modeled as vectors. It is assumed that the data association problem can be solved, i.e. if a feature is observed at two time instances, it can be recognized as the same feature. If either feature extraction or data association fails, the KF based SLAM will not be reliable.

The advantage of KF and its variants is that it provides optimal Minimum mean-square Error (MMSE) estimates of the state (robot pose and features positions), and its covariance matrix seems to converge strongly, if all the assumptions hold true. However, the Gaussian noise assumption restricts the adaptability of the KF for data association and the choice of measurement models. Besides, in EKF and the KF SLAM, because each feature is correlated to all other features, the computational cost will increase significantly during the SLAM. Eventually, computational resources will not be sufficient to update the map in real-time.

2.3.2. *Expectation maximization based SLAM approaches*

EM estimation is a statistical algorithm that was developed in the context of maximum likelihood (ML) estimation and it offers an optimal solution, being an ideal option for map-building, but not for localization. The EM algorithm is able to build a map when the robot's pose is known, for instance, by means of expectation. EM iterates two steps: an expectation step (E-step), where the posterior over robot

poses is calculated for a given map, and maximization step (M-step), in which the most likely map is calculated given these pose expectations. The final result is a series of increasingly accurate maps.

The main advantage of EM with respect to KF is that it can tackle the correspondence problem (data association problem) surprisingly well [23]. However, the need to process the same data several times to obtain the most likely map makes it inefficient. It is not efficient and can not guarantee real-time processing. When estimating the robot's pose, the cost grows exponentially with the size of the map, and the error is not bounded. Therefore after long period, the results will become unstable.

2.3.3. *Particle Filter based SLAM approaches*

Particle filters, also referred to as sequential Monte Carlo method, is a recursive Bayesian filter with particle representation. A particle set is a set of point clusters which represents the Bayesian posterior. Unlike parametric filters such as KF, PF is non-parametric and can represent non-linear, non-Gaussian model/noise. This advantage is significant since real world problems are often complicated. However, the computational cost of solving the SLAM problem by standard PF is high and impractical. As a result, PF can be applied in map-based/map-aided localization [24] but not in SLAM directly.

The development of the Rao-Blackwellized Particle Filter is a milestone for PF based SLAM. With this mathematical framework, it is possible to decompose the SLAM problem into a robot localization problem and a mapping problem which is conditioned on the robot pose estimation. This decomposition significantly reduces the computational cost of PF based SLAM and its assumption, mapping is conditionally independent given the robot trajectory, is very strong in SLAM. Therefore, the RBPF based SLAM approaches can solve SLAM problem with various mapping approaches. In FastSLAM, EKF is chosen as the mapping approach while the DP-SLAM chooses occupancy grid mapping. The success of RBPF SLAM approaches is closely related to the choice of mapping approaches, including the corresponding map representation. The map representation and the

mapping approaches can be parametric approaches or non-parametric approaches. Feature based SLAM depends on the parametric approaches, and non-parametric approaches can be used for view based SLAM which do not rely on features. The mapping approaches are introduced in Section 2.4 and the details of RBPF, FastSLAM and DP-SLAM, as well as discussions about these SLAM approaches are introduced in Section 2.5.

2.4. Map representation and environmental mapping

For mobile robot navigation, the representation of the environmental map is quite important for different tasks like SLAM and path planning. It is the foundation of environmental mapping approaches. For SLAM, mapping the environment is one of the essential requirements to complete the task. However, there are different forms of map representations and corresponding mapping approaches. Each of them has advantages and disadvantages. In general, map representations and mapping approaches can be classified into parametric and non-parametric.

The most common approach for parametric map representation is often known as the feature based approach. Features are extracted from the environment and the geometric information of the features is represented by a few parameters to represent the position of the feature [25]. Features are extracted and mapped, and they are also known as landmarks. With this information about the mapped features, the robot pose can be simultaneously updated [1]. These feature based map representations are efficient and easy to implement. However, the difficulty of extracting reliable features from unstructured, complex environments restricts the advantages.

The most common approach for a non-parametric map representation is the occupancy grid map. The basic idea of occupancy grids 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. Occupancy grid mapping algorithms implement approximate posterior estimation for those random variables. Occupancy grid maps can represent very complex, highly unstructured environments. However, computational and memory efficiency are the

drawbacks. Hence, we look for a better non-parametric map representation approach for mobile robot navigation.

2.4.1. *Feature based (parametric) map representation*

Feature extraction is the extraction of parameterized features from the source measurement data set. Most feature extractors extract a small number of features from high-dimensional sensor measurements [26]. This approach greatly reduces the data size for processing and thus reduces the computational complexity, making the following processing more efficient. The most commonly used features are point features [27].

In general, assume the two dimensional range and bearing measurement at time t is taken as $z_t = \{z_t^1, z_t^2, \dots\}$.

For each element $z_t^i = \{r_t^i, \theta_t^i, s_t^i\}$, r_t^i is the range measurement, θ_t^i is the bearing measurement and s_t^i is the signature. It is assumed that the individual measurement contains Gaussian noise and can be approximated by a Gaussian function $N(\mu, \sigma)$: μ is the bias error and σ is the standard deviation of the noise. In many applications, if the sensors are well calibrated, it is sometimes assumed that the bias error is zero. The uncertainty can be approximated as an ellipse for the bearing and range measurement as demonstrated for the 2D sensor in Figure 2-2. In 3D, the range and bearing measurement of a feature to the robot frame will be denoted as $z_t^i = \{r_t^i, \theta_t^i, \alpha_t^i, s_t^i\}$ in which the α_t^i is the pitching angle respect to the horizontal plane, as shown in Figure 2-3.

Figure 2-2: Uncertainty of point feature.

Left: Range and bearing measurement for feature L to the robot frame. Right: uncertainty for range and bearing measurement.

Figure 2-3: Range and bearing measurement for feature L in 3D

The number of features extracted at each time instance t varies according to the environmental measurement and the feature extraction algorithms implemented. However, practically, the total number of features observed at any given time instance can be treated as a constant K in the computational complexity analysis.

Feature based approaches are widely used in robot navigation. Sensor measurements are processed through feature extraction, and then the navigation algorithm is based on these extracted features. In the last two decades, feature based navigation has been extensively studied and has achieved great progress. EKF [24] and its variants such as compressed EKF [28] show the efficiency and effectiveness of SLAM. To solve the full robot posterior, FastSLAM [13] and the later version FastSLAM2 [29] have shown significant progress over EKF. Feature/landmark based SLAM is the most popular SLAM solution today.

The advantage of feature based approach is obvious. The feature extraction process ‘compressed’ the sensor information, especially for camera vision. Since vision is quite informative, it is very suitable for feature based approaches. For example, in [30], outdoor autonomous navigation is achieved with visual feature. The vehicle

can autonomously drive on the road for less than 1km in urban environment. In [31], appearance-based visual SLAM is achieved in indoor environment. In [32], small air vehicle is tested in indoor environment with camera mounted. It completes autonomous flights by detecting visual features from indoor objects. Similar work can also be found in [33] and [34]. The feature extraction not only provides localization information, it also provides the relative velocity and angular rate and can perform as visual odometry. In [35], feature based visual SLAM is applied to underwater environment. In [36], it is mentioned that feature based visual SLAM is still near real time, although I believe it can achieve in real time with latest technology.

However, feature based approaches suffer from several limitations. The first limitation is that reliable feature extraction requires certain prior knowledge about the environment. If none of this prior knowledge can be obtained, some assumption must be made about the features of the environment. When the assumption fails, the feature extraction approach fails. This is more significant in an unknown, unstructured environment. For example, data from range and bearing sensors like laser range finders and sonar are often used to extract point features. If the edges/surfaces of the objects are very smooth, such approaches will fail. This limitation is less significant for vision because vision is more informative. However, current implementations of visual SLAM take place in familiar environment with extensive experiments before real runs. It is still doubtful whether a feature extraction approach tested in indoor environment will work well in other environments like jungles. More important, vision is bearing only sensor. Even with stereo vision, the range estimation is not as accurate as laser range finders, especially at long range.

In addition, to guarantee reliable navigation results, the number of features extracted is also important. To improve localization accuracy, the number of features extracted at any time must be sufficiently large. On the other hand, too many features will cause the data association problem to be more difficult and increase the computational cost. If the environment is unknown, it is difficult to predict the number of features needed to be extracted.

With laser range finders, feature/landmark based approaches rely heavily on the success and robustness of the feature extraction algorithms. The failure of extracting features will result in the failure of SLAM. And, most importantly, to define and extract reliable features from a complicated, often unknown environment is not an easy task. In most implementations, Scale-invariant feature transform (or SIFT) [37] point features are extracted. In an urban environment with trees and lamp post, this approach can easily identify the features from the environment with a 2D laser range finder and provide good SLAM performance.

However, Figure 2-4 shows a simple environment where point feature approaches will fail with 2D laser range finder. The x-y axis of the global 2D frame is shown in red. The environment consists of smooth curved obstacles and the robot is equipped with 2D laser range finders. As long as the length of the curves is longer than the maximum range of the sensors, the robot can not extract sufficient features from the sensor measurements. The loop closing error will be very large while non-parametric approaches can still work well throughout the trajectory. Some researchers implement line features [38] which are quite common in indoor environments to mitigate this problem and these demonstrate satisfactory performance. However, in an environment consisting of curves and other complex shape edges, both line and point feature extraction algorithms would fail. These problems are due to the limitation that feature-based algorithms must have certain prior knowledge of the environment to design the feature extraction algorithms. In an environment like public parks and highways, it is easy to find features from the existing trees or lamp posts. However, it is necessary to design view-based approaches for a robust, reliable and universally-applicable robot.

Figure 2-4: Feature based SLAM in curve-like environment

The feature extraction process for bearing and range sensors will also be difficult for particular cases, especially for sonar sensors [39] [40] [41] in underwater, indoor or outdoor environment. Figure 3-4 shows a sonar image collected in the Singapore reservoir from our Blades Array sonar. The feature extraction algorithm can be implemented on the sonar image with a proper threshold to extract the underwater obstacles. However, without a prior knowledge of the obstacles and environmental conditions, these approaches are difficult to specify. Even with experiments and a number of tests, feature extraction from sonar images is still a challenging task. Non feature-based approaches, instead, do not take such a risk in the unknown environment. However, extracting features from sonar image is difficult of implementation on the data set from the Singapore reservoir, as shown in Figure 2-5. In the sonar image, the brightness of the colour represents the intensity of the sonar signal. The under water obstacles in the environment are quite unclear and most of the time, the under water objects cannot be recognized from the sonar images.

Figure 2-5: Under water sonar image

Feature extraction for sonar images is a very complicated process. The reason might be due to the non-linear sensor noise as well as the unstructured, complex, under-water environment. Without prior knowledge about the sea bed, it is difficult to determine the feature extraction method. A typical sea bed is often complicated so that neither line nor point features are reliable. Although researchers are developing feature extraction algorithms for extracting reliable features from sonar images, their performance is still subject to the environment and assumptions made. If the

sonar images can be used directly without a feature extraction process, this drawback can be avoided.

Another example is that of terrain surfaces represented by isohypse lines. They are commonly used to represent off-country terrain. It is very informative and usually enough for navigating, but it is difficult to extract features based on the complex surfaces represented unless the sensor range is long enough to detect the points A & B as shown in Figure 2-6.

Figure 2-6: Isohypse lines for terrain representation, heights in meters.

Therefore, although a feature based approach is effective and efficient for bearing and range sensors in certain environments, non feature-based approaches are important complements in a complex, unknown/unstructured environment [42]. For non-feature based mapping, non-parametric methods are needed.

2.4.2. *Non-feature based (non-parametric) mapping*

It is important to find an efficient non-parametric environmental map representation for SLAM usable with raw sensor measurement and at the same time, to implement a sophisticated sensor measurement model. An efficient non-parametric mapping approach is the foundation for an efficient view based SLAM approach. Currently, occupancy grid maps are the most common non-parametric map representations,

and their formulation is well known. However, in a grid-based approach the memory consumption is quadratic in the length in 2D space, and cubic in 3D space. Besides, the memory consumption of occupancy grid map is sensitive to the resolution. In the extreme case, if one is to exploit the accurate measurements obtained by perfect sensors, the grid resolution must be extremely high and the consequent memory requirements will be infinitely large, unless artificial noise is added to ‘waste’ sensor accuracy. In [13], robot trajectories are associated to one single occupancy grid map to greatly reduce the memory consumption and computational cost of copying multiple grid maps. However, in sparse environments, a grid map is mostly under-utilized due to the few measurement returns. In sparse 3D-space, memory inefficiency would be very significant, making these methods inapplicable.

2.4.2.1. *Occupancy grid map formulation*

The environment can be mapped based on the occupancy grid mapping approach [43]. Occupancy grid maps address 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 occupancy grids 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. Occupancy grid mapping algorithms implement approximate posterior estimation for those random variables.

The gold standard for any occupancy grid mapping algorithm is to calculate the posterior over maps given the data:

$$p(m \mid z_{1:t}, x_{1:t})$$

where m is the map, $z_{1:t}$ is the set of all measurements up to time t , and $x_{1:t}$ is the path of the robot, that is the sequence of all its poses. Taking the 2D occupancy map as an example, let m^i denote the grid cell with index i , an occupancy grid map partitions the space into finitely many grid cells:

$$m = \sum_i m^i$$

Each cell has attached to it a binary occupancy value, which specifies whether a cell is occupied or free. We will write 1 for occupied and 0 for free. The notation $p(m^i = 1)$ or $p(m^i)$ will refer to the probability that a grid cell is occupied. Based on the factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with static state. The occupancy grid mapping algorithm uses the log-odds representation of occupancy:

$$l_t^i = \log \frac{p(m^i | z_{1:t}, x_{1:t})}{1 - p(m^i | z_{1:t}, x_{1:t})}$$

The probabilities can easily be recovered from the log-odds by:

$$p(m^i | z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp(l_t^i)}$$

The update of the log-odds as new measurements are received is as follows:

$$l_t^i = l_{t-1}^i + l_{\text{sensor}}(m^i, x_t, z_t) - l_0,$$

in which l_{sensor} is the log-odds calculated from the inverse sensor model and l_0 is the constant prior of occupancy.

Compared to the grid cell resolution, the range error of the laser range finder is not significant. Therefore, the inverse sensor model can be approximated as follows: for the laser point $v(x, y, z)$ projected to the 2D grid map, if the laser point is classified as an obstacle (non-traversable), it returns l_{occupy} ; if it is classified as ground (traversable), it returns l_{free} ; if none of the laser points are projected in the cell, it returns l_0 .

Now the mapping process proceeds as follows:

If the point is classified as an obstacle, $l_t^{i,j} = l_{t-1}^{i,j} + l_{\text{occupy}}$

If the point is classified as ground, $l_t^{i,j} = l_{t-1}^{i,j} + l_{free}$

The theoretical range of the log-odds is from negative infinity to positive infinity. To efficiently record and process the log-odds, the log-odds are converted into an eight bit byte.

Figure 2-7 shows one example of a grid map generated in simulation with player/stage and orca. The black color shows the obstacles; grey is the unknown space and white is the empty/free space. The robot and the 2D laser scan are shown in blue.

Figure 2-7: An example of an occupancy grid map.

Black: detected obstacles. White: free space. Grey: unknown space.

Occupancy grid mapping has been implemented by many researchers [44] [45]. It is an effective non-parametric method for 2D map representation, but not so for 3D mapping.

2.5. Rao–Blackwellized particle filters SLAM

Doucet [8] proposed Rao–Blackwellized particle filtering for dynamic Bayesian networks. This is widely used in the robot navigation community for solving dynamic Bayesian networks.

State estimation in state-space models is widely used in a variety of computer science and engineering applications. However, the two most famous algorithms for this problem, the Kalman filter and the HMM filter, are only applicable to linear-Gaussian models and models with finite state spaces, respectively. Even when the state space is finite, it can be so large that the HMM or junction tree algorithms become too computationally expensive. This is typically the case for large discrete dynamic Bayesian networks: inference requires at each time space and time that is exponential in the number of hidden nodes.

To handle these problems, sequential Monte Carlo methods, also known as particle filters (PF), have been introduced. In the mid 1990s, several PF algorithms were proposed independently under the names of: Monte Carlo filters, sequential importance sampling with resampling bootstrap filter condensation trackers, dynamic mixture models, survival of the fittest, etc. One of the major innovations during the 1990s was the inclusion of a resampling step to avoid degeneracy problems inherent in the earlier algorithms. In the late 1990s, several statistical improvements for PFs were proposed, and some important theoretical properties were established. In addition, these algorithms were applied and tested in many domains.

One of the major drawbacks of PF is that sampling in high-dimensional spaces can be inefficient. In some cases, however, the model has a ‘tractable substructure’, which can be analytically marginalized out, conditional on certain other nodes being imputed. The analytical marginalization can be carried out using standard algorithms, such as the Kalman filter, the HMM filter, the junction tree algorithm for the general dynamic Bayesian network, or any other finite-dimensional optimal filter. The advantage of this strategy is that it can drastically reduce the size of the space over which we need to sample. Marginalizing out some of the variables is an

example of the technique called Rao–Blackwellization, because it is related to the Rao–Blackwell formula. Rao–Blackwellized particle filters have been applied in specific contexts such as mixtures of Gaussians, fixed parameter estimation, HMMs, and Dirichlet process models.

Consider the following general state space model with hidden variable R_t and observed variable Y_t . Let's assume that R_t is a Markov process of initial distribution $p(R_0)$ and transition equation $p(R_t | R_{t-1})$. The observations $Y_{1:t} \sqsubset \{Y_1, Y_2, \dots, Y_t\}$ are assumed to be conditionally independent given the process R_t of marginal distribution $p(Y_t | R_t)$. Given these observations, the inference of any subset or property of the state $R_{0:t} \sqsubset \{R_0, R_1, \dots, R_t\}$ relies on the joint posterior distribution $p(R_{0:t} | Y_{1:t})$ to estimate the distribution, or some of its characteristics such as the filtering density $p(R_t | Y_{1:t})$ or the minimum mean square error estimation $E(R_t | Y_{1:t})$.

2.5.1. Rao–Blackwellized particle filter SLAM formulation

Rao–Blackwellized particle filters are an effective means to solve the full posterior of robot pose. Each particle of $\{(x_{1:t}^{(i)}, m_t^{(i)}), i = 1, \dots, P\}$ consists of one robot trajectory $x_{1:t}^{(i)}$ and one associated environmental map $m_t^{(i)}$. To update the whole particle set $\{(x_{1:t}^{(i)}, m_t^{(i)}), i = 1, \dots, P\}$, both robot trajectory $x_{1:t}^{(i)}$ and map $m_t^{(i)}$ will be resampled (or selected) in pairs. The procedures of simultaneous localization and mapping with RBPF are shown in Figure 2-1. The key idea of the approach is to solve the recursive Bayes filter update by the equation:

$$p(x_{1:t}, m | z_{1:t}, u_{0:t-1}) =$$

$$p(m | x_{1:t}, z_{1:t}, u_{0:t-1}) p(x_{1:t} | z_{1:t}, u_{0:t-1}).$$

A particle filter is used to represent robot trajectory $x_{1:t}$ and associate it with a map m , conditioned on each sample of the particle filter. The importance weights of the

samples are therefore computed by the likelihoods of the measurements $z_{1:t}$ in the maximum likelihood map constructed by the corresponding robot trajectory. This particular particle has taken $u_{0:t-1}$ as the control.

2.5.2. *Naïve SLAM*

Naïve SLAM here stands for the SLAM approaches which:

Using a particle filter for SLAM, each particle corresponds to a specified robot trajectory through the environment and a specific occupancy grid map associated with it. By resampling the particles, both the robot trajectories and the occupancy grid maps are copied to regenerate the particles.

In [13], a naïve SLAM based on the Rao–Blackwellized particle filter is described. Each particle consists of one robot trajectory and an associated environmental map in the form of an occupancy grid. Naïve SLAM proceeds as follows:

Take sensor measurements and control inputs $z_{1:t}, u_{1:t}$

for each particle $i=1:P$;

Predict the new robot pose and update the robot trajectory based on

$$p(x_{1:t}^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t})$$

Update the occupancy grid map based on observation and new robot pose

$$p(m_t^{(i)} | x_t^{(i)}, z_{1:t}, u_{1:t})$$

for time $q=1:t$;

Calculate the particle likelihoods at time q : $L_q^{(i)} = L(x_{1:q}^{(i)}, m_t^{(i)} | z_q, u_q)$

end_for;

Calculate weights $w_t^{(i)} = \prod_{q=1}^t L_q^{(i)}$

end_for;

Resample new particle sets $(x_{1:t}^{(i)}, m_t^{(i)})$ based on weights $w_t^{(i)}$, denoted as:

$$(x_{1:t}^{(i)}, m_t^{(i)}) \sim w_t^{(i)}$$

Repeat

Here, the tilde symbol, $\sim$, denotes resampling (or selection) operations.

2.5.3. *FastSLAM*

FastSLAM was proposed in [1]. FastSLAM decomposes the SLAM problem into a robot localization problem and a collection of landmark estimation problems that are conditioned on the robot pose estimate. The factored representation is exact, due to the natural conditional independencies in the SLAM problem. FastSLAM uses a modified particle filter for estimating the posterior over robot paths. Each particle possesses K Kalman filters that estimate the K landmark locations conditioned on the path estimate. The resulting algorithm is an instance of the Rao–Blackwellized particle filter. A naïve implementation of this idea leads to an algorithm that is $O(MK)$ in the time, where M is the number of particles in the particle filter and K is the number of landmarks. They develop a tree-based structure that reduces the running time of FastSLAM to $O(M \lg K)$, making it significantly faster than existing EKF-based SLAM algorithms. The FastSLAM algorithm is also extended to situations with unknown data association and unknown number of landmarks; the approach can be extended to the full range of SLAM problems.

2.5.3.1. *SLAM problem formulation in FastSLAM*

The SLAM problem is best described as a probabilistic Markov chain. The robot's pose at time t will be denoted x_t . For a robot operating in the plane—which is the case in most of the experiments—poses are comprized of a robot's position in Cartesian coordinates and the heading direction.

Poses evolve according to a probabilistic law, often referred to as the motion model:

$$P(x_t | u_t, x_{t-1})$$

In which x_t is a probabilistic function of the robot control u_t and pose at the last time instance x_{t-1} . The motion model is usually a time invariant probabilistic generalization of robot kinematics as in [24].

The robot's environment possesses K immobile landmarks. Each landmark is characterized by its location in space, denoted θ_k for $k = 1, \dots, K$. Without loss of generality, the landmarks can be considered as points in the plane, so that locations are specified by two numerical values in 2D space.

To map its environment, the robot can sense landmarks. For example, it may be able to measure range and bearing to a landmark, relative to its local coordinate frame. The measurement at time t will be denoted as z_t . While the robot can often sense more than one landmark at a time, the sensor measurements are governed by a probabilistic law often referred to as the measurement model:

$$P(z_t | x_t, \theta, n_t) \text{ . In which } \theta = \{\theta_1, \theta_2, \dots, \theta_K\} \text{ is the set of all landmarks and } n_t \in \{1, \dots, K\} \text{ is the index of the landmark perceived at time } t \text{ .}$$

Most generally, SLAM is the problem of determining the location of all landmarks θ and the robot pose $x_{0:t}$ from measurement $z_{0:t}$ and control $u_{0:t}$, in which $0:t$ stands for the set from time 0 to the most recent time t . In probabilistic terms, this is expressed by the posterior $p(x_{0:t}, \theta | z_{0:t}, u_{0:t})$. If the correspondences are known, the SLAM problem is simpler:

$$p(x_{0:t}, \theta | z_{0:t}, u_{0:t}, n_{0:t}) \text{ .}$$

The conditional independence property of the SLAM problem implies that the posterior can be factored as follows:

$$p(x_{0:t}, \theta \mid z_{0:t}, u_{0:t}, n_{0:t}) = p(x_{0:t} \mid z_{0:t}, u_{0:t}, n_{0:t}) \prod_k p(\theta_k \mid x_{0:t}, z_{0:t}, u_{0:t}, n_{0:t})$$

The problem can be decomposed into $K + 1$ estimation problems, one problem of estimating a posterior over the robot path $x_{0:t}$, and K problems of estimating the locations of the K landmarks conditioned on the path estimate. This factorization is exact and always applicable in the SLAM problem.

The FastSLAM algorithm implements the path estimator $p(x_{0:t} \mid z_{0:t}, u_{0:t}, n_{0:t})$ using a modified particle filter. As shown below, the filter can sample efficiently from this space, providing a good approximation of the posterior even under non-linear motion kinematics. The landmark pose estimators $p(\theta_k \mid x_{0:t}, z_{0:t}, u_{0:t}, n_{0:t})$ are realized by Kalman filters, using separate filters for different landmarks. Because the landmark estimates are conditioned on the path estimate, each particle in the particle filter has its own, local landmark estimates. Thus, for M particles and K landmarks, there will be a total of $K M$ Kalman filters, each of dimension two (for the two landmark coordinates). This representation will now be discussed in detail.

2.5.3.2. *Particle filter path estimation in FastSLAM*

FastSLAM employs a particle filter for estimating the path posterior $p(x_{0:t} \mid z_{0:t}, u_{0:t}, n_{0:t})$ in (4), using a filter that is similar to the Monte Carlo localization (MCL) algorithm. MCL is an application of a particle filter to the problem of robot pose estimation (localization). At each point in time, both algorithms maintain a set of particles representing the posterior $p(x_{0:t} \mid z_{0:t}, u_{0:t}, n_{0:t})$, denoted $X_{0:t}$. Each particle $x_{0:t}^{(i)} \in X_{0:t}$ represents a guess of the robot's path. Here, the index (i) is a value between 0 and the total number of particles. The particle set $X_{0:t}$ is calculated incrementally, from the set $X_{0:t-1}$ at time instance $t - 1$, a robot control u_t , and a measurement z_t . The first particle $x_{0:t}^{(i)}$ is generated from $X_{0:t-1}$ and a probabilistic guess of the robot's pose at time t by the probabilistic motion model

$$x_{0:t}^{(i)} \sim p(x_t | u_t, x_{t-1}^{(i)}).$$

The estimate is then added to a temporary set of particles, along with the path $x_{0:t-1}^{(i)}$, under the assumption that the set of particles in $X_{0:t-1}$ is distributed according to $p(x_{0:t-1} | z_{0:t-1}, u_{0:t-1}, n_{0:t-1})$. This distribution is commonly referred to as the proposal distribution of particle filtering. After generating M particles in this way, $i = 1, 2, \dots, M$, the new set $X_{0:t}$ is obtained by sampling from the temporary particle set. Each particle $x_{0:t}^{(i)}$ is drawn with a probability proportional to a so-called importance factor or weight $w_t^{(i)}$:

$$w_t^{(i)} = \frac{p(x_{0:t}^{(i)} | z_{0:t}, u_{0:t}, n_{0:t})}{p(x_{0:t}^{(i)} | z_{0:t-1}, u_{0:t}, n_{0:t-1})}$$

The resulting sample set X_t is distributed according to an approximation to the desired pose posterior $p(x_{0:t} | z_{0:t}, u_{0:t}, n_{0:t})$, an approximation which is correct as the number of particles M goes to infinity. It is also to be noticed that only the most recent robot pose estimate $x_{0:t-1}^{(i)}$ is used when generating the particle set $X_{0:t}$.

2.5.3.3. *Landmark location estimation in FastSLAM*

FastSLAM represents the conditional landmark estimates $p(\theta_k | x_{0:t}, z_{0:t}, u_{0:t}, n_{0:t})$ by Kalman filters. Since this estimate is conditioned on the robot pose, the Kalman filters are attached to individual pose particles in $X_{0:t}$. More specifically, the full posterior over paths and landmark positions in the FastSLAM algorithm is represented by the sample set

$$X_{0:t} = \{x_{0:t}^{(i)}, \mu_1^{(i)}, \sum_1^{(i)}, \dots, \mu_K^{(i)}, \sum_K^{(i)}\}_{i=1, \dots, M}$$

where $\mu_1^{(i)}, \sum_1^{(i)}$ are the mean and covariance of the Gaussian representing k^{th} landmark θ_k , attached to the i^{th} particle. In the planar robot navigation scenario, each mean is a two element vector and the covariance is a two by two matrix.

The FastSLAM algorithm implements the update using the extended Kalman filter (EKF). As in existing EKF approaches to SLAM, this filter uses a linearized version of the perceptual model. Thus, FastSLAM's EKF is similar to the traditional EKF for SLAM in that it approximates the measurement model using a linear Gaussian function. The significant difference between the FastSLAM algorithm's Kalman filter and that of the traditional SLAM algorithm is that the updates in the FastSLAM algorithm involve only a Gaussian of dimension two, whereas in the EKF based SLAM approaches a Gaussian of size $2K+3$ has to be updated. This calculation can be done in constant time in FastSLAM, whereas it requires time quadratic in K in the standard SLAM.

2.5.3.4. *Data association for FastSLAM*

In many real world problems, the landmarks are not identifiable, and the total number of landmarks K cannot be obtained trivially. In such situations, the robot has to solve a data association problem between momentary landmarks sightings z_t and the set of landmarks in the map θ . It also has to determine if a measurement corresponds to a new, previously unseen landmark, in which case the map should be augmented accordingly.

In FastSLAM, the data association is estimated on a per particle basis: $n_t^{(i)} = \arg \max_{n_t} p(z_t | s_t^{(i)}, n_t)$. As a result, different particles may rely on different values of $n_t^{(i)}$. They might even possess different numbers of landmarks in their respective maps. This is the main difference from the EKF approaches, which determine the data association only once for each sensor measurement. It has been observed frequently that a false data association will make the conventional EKF approach fail catastrophically [24]. FastSLAM is more likely to recover, thanks to its ability to pursue multiple data associations simultaneously. Particles with a wrong data association are (in expectation) more likely to disappear in the resampling process than those that guess the data association correctly. Under mild assumptions (e.g., minimum spacing between landmarks and bounded sensor error),

the data association search can be implemented in a time logarithmic in N . The FastSLAM algorithm is an efficient new solution to the concurrent mapping and localization problem. This algorithm utilizes a Rao–Blackwellized representation of the posterior, integrating particle filter and Kalman filter representations. Similar to Murphy’s work [8], FastSLAM is based on an inherent conditional independence property of the SLAM problem. However, Murphy’s approach maintains maps using grid positions with discrete values, and therefore scales poorly with the size of the map. His approach also did not deal with the data association problem, which does not arise in the grid-based setting.

2.5.4. *FastSLAM 2.0*

FastSLAM 2.0 [29] was proposed as an improved version of the FastSLAM algorithm. The modification is conceptually simple: when proposing a new robot pose—an essential step in FastSLAM’s particle filter—the proposal distribution relies not only on the motion estimate (as is the case in FastSLAM), but also on the most recent sensor measurement. Such an approach is less wasteful with its samples than the original FastSLAM algorithm, especially in situations where the noise in motion is high relative to the measurement noise.

To obtain a suitable proposal distribution, the algorithm linearizes the motion and the measurement model in the same manner as the EKF. As a result, the proposal distribution can be calculated in closed form. This extension parallels prior work by Doucet et al., who proposed a similar modification for general particle filters [46] and Markov chain Monte Carlo techniques for neural networks [47]. It is similar to the arc reversal technique proposed for particle filters applied to Bayes networks [48], and it is similar to recent work by van der Merwe [49], who uses an unscented filtering step for generating proposal distributions that accommodate the measurement.

While this modification is conceptually simple, it has important ramifications. A convergence proof for linear SLAM problems using a single particle is also provided. The resulting algorithm requires only a constant updating time. Furthermore, it is observed experimentally that the FastSLAM2 algorithm, even

with a single particle, yields significantly more accurate results on a challenging real-world benchmark [50] than the previous version of the algorithm. These findings are of significance, as many mobile robot systems are plagued by control noise, but possess relatively accurate sensors.

FastSLAM and the improved FastSLAM 2.0 have achieved satisfactory performance in certain environments. However, like other feature based SLAM approaches, the success of FastSLAM relies on reliable feature extraction. This requires prior knowledge about the environment which is not guaranteed for unknown environments. Therefore, to navigate in unknown, unstructured complex environment, complementary view approaches are important.

2.5.5. *DP-SLAM*

Distributed particle (DP)-SLAM was proposed in [13] by Eliazar and Parr in 2003. DP-SLAM implements a simple particle filter over environmental maps and robot poses. It uses the technique called distributed particle mapping, enabling it to maintain a large number of maps efficiently, compared to naïve SLAM.

The naïve approach is doing too much work, especially in copying multiple large size occupancy grid maps. DP-SLAM implements the notion of a particle ancestry. When a particle is sampled at iteration i to produce a successor particle at iteration $i+1$, two iterations are recorded in a tree structure. Starting from a particular point, the robot trajectories would be represented by an ancestry tree. At a given time instance, suppose the laser sweeps out an area of A grid cells, where the whole map contains M grid cells, obviously for most cases, M is much greater than A , i.e. $A \ll M$.

When the problem is presented in this manner, the natural reaction from most computer scientists is to propose recording the differences between the maps, i.e. recording a list of changes that each particle makes to its map. While this would solve the problem of making efficient map updates, it would create a bad computational problem for localization: tracing a line through the map to look for an obstacle would require working through the current particle's entire ancestry and

consulting the stored list of differences for each particle in the ancestry. The complexity of this operation would be linear in the number of iterations of the particle filter. The challenge is, therefore, to provide a data structure that permits efficient updates to the map and efficient localization queries with time complexity that is independent of the number of iterations of the particle filter. The solution is through the maintaining of two data structures: the ancestry tree and the map itself.

The basic idea of the particle ancestry tree is fairly straightforward. The tree itself is rooted with an initial particle, of which all the other particles are progeny. Each particle maintains a pointer to its parent and is assigned a unique numerical ID. Finally each particle maintains a list of grid squares that it has updated.

2.5.5.1. *Map representation in DP-SLAM*

The challenge for the map representation is to devise a data structure that permits efficient updates and efficient localization. The naïve approach of a complete map for each particle is inefficient, while the somewhat less naïve approach of simply maintaining a history of each particle's update is also inefficient because it introduces a dependency on the number of iterations of the particle filter.

The solution in DP-SLAM is to associate particles with maps, instead of associating maps with particles. DP-mapping maintains just a single occupancy grid. Unlike a traditional occupancy grid, each grid square stores a balanced tree, such as a red-black tree. The tree is keyed on the IDs of the particle that have made changes to the occupancy of the square. The grid is initialized as a matrix of empty trees. When a particle makes an observation about a grid square it inserts its ID and the observation into the associated tree. Notice that this method of recording maps actually allows each particle to behave as if it has its own map. To check the value of a grid square, the particle checks each of its ancestors to find the most recent one that made an observation for that square. If no ancestor made an entry, then the particle can treat this position as being unknown.

One of the advantages about the naïve approach of keeping a complete map for each particle is the simplicity: if the cost of block copying of maps is ignored,

lookups and changes to the map can all be done in constant time. In these areas, distributed particle mapping may initially seem less efficient. Lookup on a DP-map requires a comparison between the ancestries of particle with the balanced tree at that grid square. Let D be the depth of the ancestry tree, and thus is the maximum length of a particle's ancestry. Strictly speaking, as the ancestry tree is not guaranteed to be balanced, D can be $O(P)$. However, in practice, this is almost never the case, and it is found that $D \approx O(\lg P)$, as the nature of particle resampling leads to very balanced ancestry trees.

2.5.5.2. *Computational complexity analysis*

The detailed discussion in [13] has shown that the complexity of DP-SLAM is as follows:

For a particle filter that maintains P particles, a laser that sweeps out A grid squares, and an ancestry tree of depth D , DP-SLAM requires:

$O(ADP \lg P)$ operations for localization arising from:

P Particles checking A grid squares.

A lookup cost of $O(D \lg P)$ per grid square

$O(AP \lg P)$ operations to insert new data into the tree, arising from:

P particles inserting information at A grid squares

Insertion cost of $O(\lg P)$ per new piece of information

Ancestry tree maintenance with amortized cost $O(ADP \lg P)$ arising from:

A cost of $O(\lg P)$ to remove an observation or move it up one level in the ancestry tree

A maximum potential of ADP introduced at each iteration.

2.5.5.3. *Complexity comparison*

As demonstrated in [13], DP-SLAM has an amortized complexity of $O(ADP \lg P)$, which can be as large as $O(AP \lg P)$. For the naïve approach, each map is represented explicitly, so lookups can be done in constant time per grid square. The localization step thus takes only $O(AP)$. Without the need for updates to an ancestry tree, map updates can likewise be done in $O(AP)$ time. However, the main bulk of computation lies in the need to copy over an entire map for each particle, which will require $O(MP)$ time, where M is the size of the map. Since typically $M \gg A$, this obviously is the dominant term in the computation.

DP-SLAM will be advantageous (compared to the naïve SLAM with occupancy grid maps) when $AD \lg P < M$. Even in the worst case where D approaches P , there will still be a benefit. For a high resolution occupancy grid, M will be quite large since it grows quadratically with linear measurement of the 2D environment and would grow cubically in a 3D environment. Moreover, A , will be a tiny fraction of M . The size of $D \lg P$ will, of course, depend upon the environment features and the sampling rate of the data.

DP-SLAM associates the map to the robot path. This approach significantly reduces the computational cost for solving the SLAM problem. The complexity analysis shows its advantages. However, the number of grid cells swept by the laser beam, A , is unclear. While the robot is moving around, the distance travelled by the robot might far exceed the maximum range of the laser range finder or any other sensing devices. Therefore, A_t , the number of grids swept by the laser beam at certain time instance t , is different from $A_{0:t}$, the overall number of grids swept by the laser beam from the starting time.

2.5.5.4. *Discussion of DP-SLAM*

From the descriptions of DP-SLAM approaches and SLAM results in [13], DP-SLAM combines the occupancy grid map with the RBPF-SLAM approach more efficiently than the naïve SLAM. DP-SLAM demonstrates that the method of a

non-parametric map representation can be applied to the RBPF-SLAM framework and perform well.

From DP-SLAM, several key points can be noticed.

Firstly, non-parametric methods can be used for map representation under an RBPF-SLAM framework. Feature extraction and landmark mapping is not necessary for DP-SLAM.

Secondly, the empty (free) space and the unknown space of the occupancy grid map are not significant for SLAM results. They can be ignored to improve efficiency.

Thirdly, the map representation plays an important role in computational efficiency. However, the grid map is not suitable for 3D space. It requires a very large memory to represent a 3D environmental space in high resolution.

Fourthly, the area ‘swept’ by the sensor depends on the measurement noise. If the cell size (the length and width of the cell) of the occupancy grid map is much smaller than the measurement noise, then A , the number of grid cells ‘swept’ by sensor, will still be large. Especially, when the sensor is not accurate, this problem will become more significant.

Fifthly, the computational cost for each occupancy grid cell in DP-SLAM is not balanced. Only the cells ‘swept’ by the sensors require processing and the rest do not. This will not increase computational cost significantly but it is not suitable for parallel computing. In addition, the tree structure used in DP-SLAM further increases this difficulty.

Finally, DP-SLAM is only ‘near real time’ for simple corridor like environments as described in [13]. The updating time of DP-SLAM testing takes a few seconds. For more complicated scenarios, the processing speed of DP-SLAM is not suitable for real time navigation.

Therefore, although DP-SLAM is capable of implementing non-parametric methods in a RBPF-SLAM framework, it is still not the ideal solution for view based SLAM. Further development and new approaches are needed.

2.6. *Summary*

To localize the robot accurately is one of the most important requirements for mobile robot navigation. An odometry/INS system can provide localization information but the error is not bounded. Therefore, it is not accurate enough for mobile robot navigation. Although map-based/map-aided localization can provide accurate localization results with bounded errors, it requires a pre-available environmental map. Simultaneous localization and mapping (SLAM), on the other hand, localizes the robot and builds the environmental map at the same time without a pre-available environmental map. Therefore, our research focuses on SLAM for mobile robots. Rao–Blackwellized particle filter SLAM has been proven in the last decade to be an efficient and robust SLAM approach. The mapping approaches and the corresponding map representation methods are important for RBPF based SLAM. Feature based mapping using parametric methods is not suitable in a complex, unstructured unknown environment. The occupancy grid mapping using non-parametric methods are not efficient, especially not appropriate in 3D mapping. As a result, the existing view based RBPF-SLAM approaches are not efficient enough to navigate in a complex, unstructured, unknown environment. Further improvements are still needed.

CHAPTER III: GENETIC ALGORITHMIC FILTER MAPPING

3.1. *Motivation*

The success of mapping approaches and the corresponding map representation play an important role for RBPF based SLAM. The feature based (parametric) mapping approaches rely on the prior knowledge about the unknown environment. This limitation restricts its use in highly complex, unstructured environment. Occupancy grid mapping is the common non-parametric mapping approach exploited in non-feature based mapping, though it is computationally expensive.

An alternative to using a grid for non-parametric map representation is a particle map. It is more efficient than occupancy grids, especially in mapping 3D environments. However, the particle map representation suffers from the sample impoverishment problem in the re-sampling process and this problem is difficult to solve by analytic and simple heuristic means. This problem is mitigated in our development of an efficient Genetic Algorithmic Filter (GAF) mapping for view based SLAM approaches. The strategy and the resulting GAF is very effective and efficient, and especially appropriate for 3D mapping.

In this chapter, Section 3.2 introduces the particle map representation and the compressed particle filter mapping. The proposed GAF mapping is described in Section 3.3. The simulation and experimental results which compared the mapping approaches are shown in Section 3.4 and Section 3.5, followed by a summary in Section 3.6.

3.2. *Compressed particle filter mapping*

3.2.1. *Particle filters*

In particle filters, the posterior distribution is represented using samples known as

particles and is denoted as $X_t := x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[N]}$. Each particle $x_t^{[i]}$ ($1 \leq i \leq N$) is a concrete instantiation of the state (robot pose) at time t , a hypothesis as to what the true state may be at time t . The constant N denotes the number of particles in the particle set X_t . The intuition behind particle filters is to approximate the belief $bel(x_t)$ by the set of particles X_t . The likelihood for a state hypothesis x_t to be included in the particle set X_t is proportional to its Bayesian filter posterior $bel(x_t)$. The input of this algorithm is the particle set X_{t-1} , along with the most recent control u_t and the most recent measurement z_t . For map-aided localization, if the environment map m_t is always available, the robot pose's posterior at time t is thus

$$\overline{bel(x_t)} = \int p(x_t | u_t, x_{t-1}, m_t) bel(x_{t-1}) dx$$

$$bel(x_t) = \eta p(z_t | x_t, m_t) \overline{bel(x_t)}.$$

The updating process of the particle filter algorithm is

At time t , initialize the particle sets $\bar{X}_t = X_t = \emptyset$.

For all particles, $i = 1, \dots, N$.

{

Sample: $x_t^{[i]} \sim p(x_t | u_t, x_{t-1}^{[i]})$,

Calculate weights: $w_t^{[i]} = p(z_t | x_t^{[i]})$

$\bar{X}_t = \bar{X}_t + \{x_t^{[i]}, w_t^{[i]}\}$.

}

Resampling select $x_t^{[i]} \propto w_t^{[i]}, i = 1, \dots, N$.

Add $x_t^{[i]}$ into X_t .

More detailed information on particle filters for mobile robot navigation can be found in [14]. One example of particle filter localization can be found in [51].

3.2.2. *Particle map and compressed particle map*

If the environmental map is represented by particles, they are called map particles. The environmental map represented by map particles can be called a particle map. It would be capable of implementing resampling processes over the map particles to mitigating the sample depletion problem. The global map is approximated by a number of particles m_i with a total of k map particles.

$$p(m \mid z_{1:t}, x_{1:t})$$

$$m = \sum_{j=1}^k m^j$$

Due to the sample depletion problem, the particle representation performs less efficiently in a large space with very low probability density. On the other hand, the performance of the grid map representation is not affected by the ‘shape’ of belief. If the belief is almost evenly distributed in the space, the grid map representation will perform better than a particle map representation. For the occupancy grid map, it is often assumed that the belief is over the entire navigation space. However, for SLAM, this is not the case. In DP-SLAM [13], although the grid map representation is implemented, the actual space processed in the SLAM algorithm is only the space ‘swept’ by the laser range finders. Or say, the 1D edge of the obstacles detected by 2D laser range finders or the 2D surface of the obstacles detected by 3D laser range finders. The space processed for the SLAM algorithm is one dimension less than the entire space. In addition, the more accurate the sensor it is, the more ‘concentrated’ the belief would be. Therefore, in such a case, it is not necessary for particle map representations to sample the entire space.

In addition, a sensor measurement contains measurement noise. After a number of

measurements over a period of time, the uncertainty of the obstacle can be reduced. The particle map representation may have less efficiency in the beginning, but after resampling process over a certain time period, the map belief ‘converges’ and the approximation errors will be greatly reduced [12]. Therefore, the particle map representation can achieve good efficiency in environmental map representation for mobile robot navigation.

However, there are limitations for the particle map representation.

Firstly, there are new areas of space observed while the robot is moving, so new map particles must be created continuously. However, it is hard to determine how many new map particles should be created. When the robot is stationary and new map particles are created all the time, the computational cost will be unbounded. Therefore, the number of map particles must be controlled, and it should increase according to the number of obstacles detected, not the space explored by the robot or the time elapsed during the navigation. When the robot is stationary, the number of map particles should remain a constant.

Our solution is to use the method of compressed map particles. Map particles are constantly created according to the obstacles detected (sensor measurements). However, when new map particles are created, they are evaluated based on the existing map particles. With a virtual particle size or desired map resolution ε , if two map particles are very near to each other, they are compressed into one compressed particle with a count value C . In this way, one compressed map particle can approximate C map particles. Therefore, the number of compressed map particles will be proportional to the obstacles observed and independent of the space explored.

The compression process takes computational resources. However, as will be shown in Chapter IV, for the Rao–Blackwellized Particle Filter SLAM, the computational cost for such particle map compression is negligible compared to the computational cost for resampling the Rao–Blackwellized particles.

The major drawback of the compressed particle map lies in the resampling stage.

The standard resampling process implements Monte Carlo resampling over the particle set. However, when such an approach is applied to the compressed particle map, the environmental map would converge to a few particles and provide an erroneous map representation. Heuristic methods have been developed to overcome this problem. For example, adaptive resampling [10] [52] examines the particle diversity to control the resampling rates. Other approaches like artificial noise can be implemented as well. However, these solutions have not been developed in a systematic manner. To mitigate the shortcomings of the compressed particle filter, a genetic algorithm filter is proposed. It makes use of the highly developed genetic algorithm techniques and concepts to improve the compressed particle filter.

3.3. Genetic algorithmic filter mapping

Genetic algorithms mimic the Darwinian selection of the fittest found in nature [53]. It uses a form of guided randomness, partly based on heuristics, to search a complex solution space. It quickly converges to a good approximation of the solution, is general enough to be applied to any search space, and is very robust. Genetic algorithms are different from other heuristic methods in that they work on a population of possible solutions, while other heuristic methods using a single solution in their iterations are probabilistic (stochastic), not deterministic. It has been introduced to feature based mapping [26] and SLAM [27] [28].

Genetic algorithms use natural selection as their paradigm for finding better solutions. Briefly, a candidate solution to a problem is encoded as a string, and a starting population of solutions is created randomly. The population of candidates is then evolved over generations through the application of genetic operators. In each generation the candidates are each assigned a fitness rating according to some criteria (objective function), and then that rating is scaled into a probability of reproduction. The candidates are then recombined to produce the next generation. The best-fit solution is then chosen as the solution to the problem. The concept of genetic algorithm can be illustrated as shown in Figure 3-1 below.

Figure 3-1: Illustration of a genetic algorithm

In this section, the techniques of genetic algorithms are used to mitigate the problems of compressed particle filters, resulting in the genetic algorithmic filter (GAF) approach. To apply the genetic algorithmic filter approach to the environmental map representation, there are several concerns that must be addressed.

- The format of the genetic codes to form the chromosomes
- The choice of fitness function for evaluation
- The choice of selection function

The advantage of applying GA is that it has been highly developed with many sophisticated techniques. These techniques have been proven by applications in other research areas; the plusses and minuses of each technique are well known. To

choose the proper techniques, the divergence and convergence of different techniques must be balanced.

The map chromosomes are generated according to the observations, the measurement noise and the robot pose. At the beginning, a number of chromosomes are generated and during the movement of the robot, some chromosomes are incrementally added at each time interval. Simple Monte Carlo resampling would not be capable of representing this distribution efficiently. It is possible that a few ‘very fit’ chromosomes may dominate the whole chromosome set, resulting in a convergence into only a few chromosome clusters and, finally, one point. To mitigate this so-called sample impoverishment problem, mutation and crossover operations are not enough to provide sufficient spread to the chromosome set. Therefore, a parallel version of the genetic algorithm known as the island model [54] is implemented. The map chromosomes are separated into groups called ‘islands’. Besides, tournament selection in GA proves that it can control the selection force during the re-sampling. It is chosen to replace Monte Carlo selection in each of the islands in the parallel evolution of the chromosome populations. In this section (3.3), the details of 2D GAF mapping will be presented, and they can easily be extended to 3D GAF mapping.

However, GAF mapping is obtained by integrating genetic algorithm with particle filters. Therefore, it has several advantages that GA doesn’t have. The Figure 3-2 shows the procedures of GAF.

Figure 3-2: Illustration of a genetic algorithm

There are several major differences between standard GA and the proposed GAF approach. In GA, the evolution keeps running and only stops when a satisfactory solution is found. Since the time to complete GA evolution is unknown, it is not considered as real time algorithm. However, in GAF, each evolution cycle only take place once for a time interval. As the number of chromosomes is fixed and finite, the update time of GAF can be estimated with an upper bound.

In GA, the goal of evolution process is to find out a final optimized or sub-optimized solution. However, GAF is aim to update and maintain the whole chromosome set and the only stopping condition is the termination of SLAM.

In GA, the fitness function is directly set by developers. However, the evaluation process (fitness function) in GAF is based on the sensor models.

In GA, the number of chromosome is usually fixed as a constant. However, in GAF, it varies during the navigation. New chromosomes are generated in every time interval. As a result, the number of chromosome in GAF keeps increasing and proportional to the number of obstacles/objects observed.

The following sections describe the proposed GAF approach in details.

3.3.1. Basic notations

2D Cartesian coordinates are used for the 2D GAF mapping approach as shown in Figure 3-3. The two axes, $\vec{a}$ and $\vec{b}$, are normal to each other and any point in such 2D Cartesian coordinates is represented as (a,b) . The heading angle θ is counter-clockwise, oriented from $\vec{a}$ towards $\vec{b}$. All distances and locations are in meters and the heading angle is represented in radians.

Figure 3-3: 2D Cartesian coordinates

In this and following chapters, one example for the general format of variable notation is presented in Figure 3-4. The format is used to represent robot pose/trajectory, environmental map and map chromosomes, etc.

Figure 3-4: 2D Cartesian coordinates

Let's take variable notation ${}^r A_t^{(i)}$ as an example. The subscript t will be the time index. The robot navigation starts at time $t=0$ and increase by 1 for every interval. If the subscript is $0:t$, it indicates a sequences of variable state from the start to time t , i.e. ${}^r A_{0:t}^{(i)} = \{{}^r A_0^{(i)}, {}^r A_1^{(i)}, {}^r A_2^{(i)}, \dots, {}^r A_t^{(i)}\}$. The superscript (i) is its index ID among a set, i.e. ${}^r A_t = \{{}^r A_t^{(0)}, {}^r A_t^{(1)}, {}^r A_t^{(2)}, \dots\}$. If necessary, the sub-name is marked at the top left to distinguish different variables.

Measurement models describe the formation process by which sensor measurements are generated in the physical world. If there are K_s individual measurements in z_t , we can approximate $p(z_t | m_t, x_t)$ by the product of the individual measurement likelihoods

$$p(z_t | m_t, x_t) = \prod_{k=1}^{K_s} p(z_t^{(k)} | m_t, x_t)$$

With environment map m_{t-1} , robot pose x_t and sensor measurement z_t , the new map posterior m_t is estimated by the inverse measurement model $p(m_t | z_t, x_t)$.

GAF mapping is capable of adapting arbitrary measurement models and inverse measurement models. The models described in this chapter are examples for the convenience of presenting GAF mapping.

3.3.2. Genetic coding: chromosome map representation

The typical format of a genetic code is a binary code. The properties of the elements are encoded into 1s and 0s. However, such a representation is not suitable for environmental map representations. The environment map is represented by a set of map particles, here, referred to as map chromosomes. The properties of each map chromosome consist of its position (2D or 3D) in global Cartesian coordinates and the count value of the compressed particle. Therefore, the genetic code of the map chromosome is consists of its position together with a count value which is used in chromosome compression.

As an example, in 2D coordinates, each map chromosome (particle) represents occupancy. A number of map chromosomes form a map hypothesis at a certain time t . The location of a single map chromosome in 2D Cartesian coordinate is represented as $(^m a, ^m b)$ where there is no orientation involved. Therefore, the genetic code of a single map chromosome is then $(^m a, ^m b, ^m Count)$, in which the $^m Count$ is the count value of that chromosome. In our approach, the environmental map hypothesis is approximated by a number of compressed map chromosomes. The environment map can also be considered as being represented by chromosome set. If a map consists of K chromosomes, the chromosome ID is $j = 1, 2, \dots, K$. The genetic code of single map chromosomes can be represented as

$m_t^{(j)} = (^m a_t^{(j)}, ^m b_t^{(j)}, ^m Count_t^{(j)}), j = 1, 2, \dots, K$, if there are multiple maps to be represented and the map ID is $i = 1, 2, \dots, P$.

The map chromosomes at time t will be represented as:

$$m_t^{(i,j)} = (^m a_t^{(i,j)}, ^m b_t^{(i,j)}, ^m Count_t^{(i,j)}), i = 1, 2, \dots, P; j = 1, 2, \dots, K$$

For single map mapping, of course, then we have $P = 1$.

Weights (fitness value) are calculated for the map chromosome in the map evaluation stage.

Each map chromosome (particle) set approximates the hypothesis of the true map posterior at the time instance t .

$$m_t^{(i)} \approx \bigcup_{j=1}^K m_t^{(i,j)}$$

For multiple maps with map ID $i = 1, 2, \dots, P$, all the map hypotheses approximate the multiple map posterior at time t .

$$m_t \approx \bigcup_{i=1}^P m_t^{(i)}$$

3.3.3. *Chromosome map compression*

The compressed map chromosome is always combined with a parameter called the count value ${}^m\text{Count}_t^{(i,j)}$. The compressed map chromosome is virtually a circle in 2D space or a sphere in 3D space. Any other map chromosome located in the circle (sphere) will be considered as the same map chromosome. Let the previous map chromosome be $j1$ and the newly generated one be $j2$. A minimum radius ε is chosen which determines the resolution of the chromosome map. The smaller the ε , the higher the final environmental map resolution achieved.

$$m_t^{(i,j1)} = ({}^m a_t^{(i,j1)}, {}^m b_t^{(i,j1)}, {}^m\text{Count}_t^{(i,j1)})$$

$$m_t^{(i,j2)} = ({}^m a_t^{(i,j2)}, {}^m b_t^{(i,j2)}, {}^m\text{Count}_t^{(i,j2)})$$

The compression is done such that the new map chromosome will not be generated; instead the count value of the old map particle $j1$ will be increased.

If $|{}^m a_t^{(i,j1)} - {}^m a_t^{(i,j2)}| < \varepsilon$ or $|{}^m b_t^{(i,j1)} - {}^m b_t^{(i,j2)}| < \varepsilon$, then

${}^m\text{Count}_t^{(i,j1)} = {}^m\text{Count}_t^{(i,j1)} + {}^m\text{Count}_t^{(i,j2)}$ and the $m_t^{(i,j2)}$ will be deleted.

3.3.4. *Map chromosomes generation*

For a chromosome map $m_t^{(i)}$, map chromosomes are generated based on sensor measurements z_t and robot pose $x_t^{(i)}$. The sensor measurement z_t of the 2D laser

range finder usually consists of a number of range values associated with the relative bearing angle. For example, let the sensor field of view be 180 degrees and the angular resolution be 0.5 degree, there would be 361 range values obtained for a single laser scan.

$$\begin{aligned}\tau_t^{(n)} &= (n-180) * \pi / 180 \\ z_t^{(n)} &= (r_t^{(n)}, \tau_t^{(n)}) \\ z_t &= \bigcup_{n=0}^{360} z_t^{(n)}\end{aligned}$$

Here, $\tau_t^{(n)}$ is the bearing angle for a single range value in the direction of the robot heading, assuming the laser range finder is mounted facing forward on the robot. $r_t^{(n)}$ is the range measurement for the particular laser beam, n is the index between 0 and 360. The final laser range measurement therefore consists of 361 range measurements and 361 associated bearing angles, $z_t = \bigcup_{n=0}^{360} z_t^{(n)}$.

In the sensor coordinates as shown below, each laser range beam can provide a single point in 2D space, indicating an obstacle $o_t^{(n)}$ occupying the space.

$$\begin{aligned}o_t^{(n)} &= ({}^o a_t^{(n)}, {}^o b_t^{(n)}) \\ {}^o a_t^{(n)} &= r_t^{(n)} * \cos(\tau_t^{(n)}) \\ {}^o b_t^{(n)} &= r_t^{(n)} * \sin(\tau_t^{(n)})\end{aligned}$$

However, a sensor measurement usually comes with certain sensor measurement errors. Assuming the sensor measurement error is Gaussian distributed in the range and the bearing, then:

$$\begin{aligned}o_t^{(n)} &= ({}^o a_t^{(n)}, {}^o b_t^{(n)}) \\ {}^o a_t^{(n)} &= (r_t^{(n)} + N(0, \sigma_{srange})) * \cos(\tau_t^{(n)} + N(0, \sigma_{sbearing})) \\ {}^o b_t^{(n)} &= (r_t^{(n)} + N(0, \sigma_{srange})) * \sin(\tau_t^{(n)} + N(0, \sigma_{sbearing}))\end{aligned}$$

If the laser range finder is not mounted on the centre of the robot, a frame transformation is required to obtain the correct position of the obstacle. Let the sensor be mounted on the robot at the position $x_t^{(i)} = ({}^s a, {}^s b, {}^s \theta)$ at position $({}^s a, {}^s b)$

and heading ${}^s\theta$, then the position of the obstacle with respect to the robot frame would be approximated as

$$\begin{aligned} {}^r a_t^{(n)} &= {}^s a + {}^o a_t^{(n)} * \cos({}^s\theta) + {}^o b_t^{(n)} * \sin({}^s\theta) \\ {}^r b_t^{(n)} &= {}^s b + {}^o a_t^{(n)} * \sin({}^s\theta) - {}^o b_t^{(n)} * \cos({}^s\theta) \end{aligned}$$

In the global frame, if the robot pose is $x_t^{(i)} = (a_t^{(i)}, b_t^{(i)}, \theta_t^{(i)})$, then the position of the obstacle with respect to the global frame would be $({}^m a_t, {}^m b_t)$

$$\begin{aligned} {}^m a_t^{(n)} &= a_t^{(i)} + {}^r a_t^{(n)} * \cos(\theta_t^{(i)}) + {}^r b_t^{(n)} * \sin(\theta_t^{(i)}) \\ {}^m b_t^{(n)} &= b_t^{(i)} + {}^r a_t^{(n)} * \sin(\theta_t^{(i)}) - {}^r b_t^{(n)} * \cos(\theta_t^{(i)}) \\ n &= 0, 2, \dots, 360 \end{aligned}$$

The new map particles (chromosomes) are generated according to the $({}^m a_t, {}^m b_t)$. After the new map particles are generated, they are processed and compressed (see Section 3.3.2) with the existing map chromosomes. The map chromosomes that can not be compressed are added into the chromosomes map.

Generate K_t chromosomes where the number $K_t \propto \beta * n_{z_t}$. n_{z_t} is the number of observations at time t and β is a constant chosen according to the computational resources. With the laser range finder measurement z , each laser beam returns a range measurement with the bearing information attachment.

There are sophisticated measurement models and inverse measurement models. GAF mapping has no restriction on the model used. A model should give the weight based on $p(m_t^{(i)} | x_t^{(i)}, z_t)$. Here, examples of sensor models are presented, used to evaluate and generate the map chromosomes.

Let the 1D sensor measurement model function which gives the importance weight be $M(R, l)$ (as presented in section 3.4.3), R is the sensor range measurement and l is the distance between the sensor and certain position in the 1D space. For a certain position in the 1D space, the higher the weight, the higher the probability of the occupancy.

Based on the 1D sensor measurement model, our direct implementation of the approximation is simplified as in Figure 3-5. The uncertainty of the range measurement is σ_{range} .

Figure 3-5: Illustration of 1D range sensor model

For a certain point in the 1D sensor measurement model, if the distance l is much less than the measured range, it is considered as empty space where there are no obstacles blocking the laser range finder.

$$l < r - \sigma_{range}$$

If the distance is far beyond the measured range, it is considered as unknown space where no conclusion can be made.

$$l > r + \sigma_{range}$$

If the distance is roughly the measurement range, it is considered an obstacle space. New map particles might be generated there.

$$r - \sigma_{range} \leq l \leq r + \sigma_{range}$$

The approximation of the measurement model is to approximate it by discrete 1D grids with the resolution ε .

A minimum threshold χ is set for map chromosome generation. If the weight is less than the threshold χ , no chromosome is generated. If the weight is higher than χ , a new chromosome should be generated. If the weight is much higher than χ , only one new map chromosome is generated. However, the count value ${}^m\text{Count}$ is set to be $M(R, l) / \chi$. The decision as to whether to generate a new map

chromosome can be summarized as below, for $l \in (0, range_{\max})$ in the increment of ε

$$\left\{ \begin{array}{l} l < r - \sigma_{srange} \longrightarrow No \\ l > r + \sigma_{srange} \longrightarrow No \\ r - \sigma_{srange} \leq l \leq r + \sigma_{srange} \longrightarrow \left\{ \begin{array}{l} 0 < M(r, l) < \chi \longrightarrow No \\ M(r, l) > \chi \longrightarrow Yes, mc = M(R, l) / \chi \end{array} \right. \end{array} \right.$$

In 2D space, the condition to determine map chromosome generation is similar to the 1D case. However, both range and bearing measurement error must be taken into account. Let the weight function (measurement model) for the 2D case be $M(r, \alpha, l, h)$ in the sensor's polar coordinates. R is the sensor measurement range, α is the sensor beam heading, l is the distance between the sensor and a position in the 2D space, and h is the bearing from the sensor to the position.

The new map chromosomes are generated with the minimum resolution ε in 2D space. If the position is in empty space or unknown space, no the new map particles will be generated there. Let the position in 2D space be $({}^m a, {}^m b)$, the relative range and the relative bearing to the sensor polar coordinates will be

$$l = \sqrt{{}^m a * {}^m a + {}^m b * {}^m b}$$

$$h = \arctan({}^m b, {}^m a)$$

For all $({}^m a, {}^m b)$ in the increment of ε , a new map chromosome will be generated if

$$\left\{ \begin{array}{l} M(r, \alpha, l, h) < \chi \longrightarrow No \\ M(r, \alpha, l, h) > \chi \longrightarrow Yes, mc = M(R, l) / \chi \end{array} \right.$$

The new map chromosome generated at $({}^m a, {}^m b)$ with the count value ${}^m Count$ will be transformed into the global frame.

3.3.5. *The fitness function: chromosome evaluation*

The purpose of the fitness function in GA is to evaluate the ‘goodness’ of the chromosomes. In GAF, mapping is based on the sensor measurements, e.g. a laser range finder. Therefore, it is natural to design the fitness function based on sensor measurements and calculate the ‘goodness’ of the chromosomes.

The map chromosome is evaluated according to the sensor measurement z and the robot pose x . The sensor measurement model is as shown in Section 3.3.3. The weight is calculated according to measurement model. The weight of the map chromosomes at time t is:

$$^m w_t^{(i,j)} \propto p(z_t | x_t^{(i)}, m_{t-1}^{(i,j)})$$

Or, the fitness function in GAF is thus:

$$f_{mp}^{(i,j)}(m_t^{(i,j)}, x_t^{(i)}, z_t) = ^m w_t^{(i,j)}$$

Let’s take the 2D measurement model described in 3.3.3 as an example and formulate it in detail. If the position of the existing map particle in the 2D space is $m_t^{(i,j)} = (^m a_t^{(i,j)}, ^m b_t^{(i,j)})$, the relative range and the relative bearing to the sensor polar coordinates will be

$$l_t^{(i,j)} = \sqrt{{}^m a_t^{(i,j)} * {}^m a_t^{(i,j)} + {}^m b_t^{(i,j)} * {}^m b_t^{(i,j)}}$$

$$h_t^{(i,j)} = \arctan({}^m b_t^{(i,j)}, {}^m a_t^{(i,j)}).$$

The weight (fitness value) with sensor measurement $z_t^{(n)} = (r_t^{(n)}, \alpha_t^{(n)}), n = 0, 1, 2, \dots, 360$ obtained, will be

$$f_{map}^{(i,j)}(m_t^{(i,j)}, x_t^{(i)}, z_t) = {}^m w_t^{(i,j)} = V(z_t, l_t^{(i,j)}, h_t^{(i,j)}) = \sum_{n=0}^{360} M(r_t^{(n)}, \alpha_t^{(n)}, l_t^{(i,j)}, h_t^{(i,j)})$$

Since it is unlikely for a map chromosome to be observed by many beams, the fitness function can be simplified as:

$$f_{map}^{(i,j)}(m_t^{(i,j)}, x_t^{(i)}, z_t) = {}^m w_t^{(i,j)} = V(z_t, l_t^{(i,j)}, h_t^{(i,j)}) = M(r_t^{(n)}, \alpha_t^{(n)}, l_t^{(i,j)}, h_t^{(i,j)})$$

3.3.6. *Island model*

The purpose of applying the island model is to restrict the area affected by ‘very fit’ chromosomes’ hijacking the entire population of chromosomes. The chromosomes are arranged in groups according to their position. In the island (group) $G(col, row)$, the chromosomes $m_t^{(i,j)} = ({}^m a, {}^m b, {}^m Count)$ have the property of:

$$|{}^m a / L| = col \text{ and } |{}^m b / L| = row$$

Here, the $| \cdot |$ operator represents the integer round and L is a constant value greater than zero, approximately the same as the maximum sensor range. Therefore, all the chromosomes in $G(col, row)$ are in the same $L \times L$ square (island). The number of chromosomes in island groups may vary. The selection process is then implemented inside an island.

3.3.7. *The selection: map updating*

As shown in Section 3.3.4, the weight of a map chromosome

$$V(z_t, l_t^{(i,j)}, h_t^{(i,j)}) = \sum_{n=0}^{361} M(r_t^{(n)}, \alpha_t^{(n)}, l_t^{(i,j)}, h_t^{(i,j)}), j = 1, 2, \dots, K$$

is evaluated based on its position $m_t^{(i,j)}$ and corresponding robot pose $x_t^{(i)}$ with sensor measurement z_t at time t . The total number of chromosomes is reduced after the new map chromosomes are generated by the chromosome compression.

To achieve high efficiency in the map representation, a resampling process is taken on all the map chromosomes (particles). In GAF-mapping, tournament selection is used and performs well compared to the normal Monte Carlo selection. The purpose of applying tournament selection is to restrict the number of chromosomes affected by ‘very fit’ chromosomes. In this thesis, tournament selection is done within one group of chromosomes. If no chromosomes are in that group, no selection is needed. The environmental map chromosome updating is shown as below:

Let the random function $rand(X,Y)$ generate a random integer between the integers X and Y . Then draw two integers randomly in $(0,K)$

$$j1 = rand(0, K)$$

$$j2 = rand(0, K)$$

- If both map chromosome $m_t^{(i,j1)}$ and $m_t^{(i,j2)}$ are in the field of view at time t , then the map chromosomes $m_t^{(i,j1)}$ and $m_t^{(i,j2)}$ will compete with each other and kill the loser according to their weight values.
- If $V(z_t, l_t^{(i,j1)}, h_t^{(i,j1)}) > V(z_t, l_t^{(i,j2)}, h_t^{(i,j2)})$, then chromosome $j1$ wins. The count value of $j2$ will be added to the count value of $j1$, and $j2$ will be deleted.
- If $V(z_t, l_t^{(i,j1)}, h_t^{(i,j1)}) < V(z_t, l_t^{(i,j2)}, h_t^{(i,j2)})$, then chromosome $j2$ wins. The count value of $j1$ will be added to the count value of $j2$, and $j1$ will be deleted.
- If $V(z_t, l_t^{(i,j1)}, h_t^{(i,j1)}) = V(z_t, l_t^{(i,j2)}, h_t^{(i,j2)})$, then none of the map chromosomes are deleted, and no action is required.

In summary, the tournament selection is designed as follows:

Draw two chromosomes $m_t^{(a)}, m_t^{(b)}$ from the same island,

Compare the two weights calculated,

If $w_t^{(a)} > w_t^{(b)}$, then chromosome a will replace chromosome b, $m_t^{[a]} \rightarrow m_t^{[b]}$.

Else chromosome b will replace chromosome a, $m_t^{[a]} \leftarrow m_t^{[b]}$.

Repeat for all the chromosomes.

3.3.8. *Memory operation for chromosome generation and deleting*

The map chromosomes involve a lot of chromosome generation and deleting operations. Due to the advantage of the chromosome (particle) representation, the sequence of the chromosomes is not relevant for the mapping results. Therefore, to ease hardware implementation, the chromosome generation and deleting operations can be simplified and become very efficient.

Let the number of map chromosomes be $K_t^{(i)}$. This variable varies from time to time. To generate a new map chromosome, increase $K_t^{(i)}$ by one and add the new map chromosome as $m_t^{(i,K)}$, i.e., appending the new map chromosome to the end of the array. To delete the old map chromosome j, copy $m_t^{(i,j)} = m_t^{(i,K)}$ and decrease $K_t^{(i)}$ by one, i.e., copy the last element to overwrite the map chromosome.

3.3.9. *Crossover and mutation*

Let's denote the crossover function as:

$$crossover(m_t^{(i,j)}, pc), j = 1, 2, 3, \dots, K.$$

In which pc is the crossover rate, the probability for the crossover operation to be processed.

Draw random number $pr \in (0,1)$.

If $pr < pc$,

{

Generate random integer $A \in (0, K)$.

$$crossover : m_t^{(i,A)} \square m_t^{(i,j)} \}$$

}

Repeat all.

The mutation function is represented as $Mutation(m_t^{(i,j)}, R, pm)$, in which, pm is the mutation rate and R is the mutation range for chromosomes.

Draw random number $pr \in (0,1)$.

If $pr < pm$, for $m_t^{(i,j)} = ({}^ma, {}^mb, {}^mCount)$, then
$$\begin{aligned} {}^ma &= {}^ma + N(0, R) \\ {}^mb &= {}^mb + N(0, R) \end{aligned}$$
.

3.3.10. Summary of GAF mapping

The GAF mapping can then be summarized as the following steps in pseudo-code:

Process_GAF_mapping($m_{t-1}^{(i)}, x_t^{(i)}, z_t$)

{

Evaluation: $f_{map}^{(i,j)}(m_{t-1}^{(i,j)}, x_t^{(i)}, z_t) = {}^mw_t^{(i,j)} \propto p(z_t | x_t^{(i)}, m_{t-1}^{(i,j)})$ for all map chromosomes.

Selection for all map chromosomes: $\widehat{m}_{t-1}^{(i)} = Selection(m_t^{(i)})$

For all map chromosomes $Crossover(m_{t-1}^{(i,j)}, pc), j = 1, 2, \dots, K$

For all map chromosomes $Mutation(m_{t-1}^{(i,j)}, r, pm), j = 1, 2, \dots, K$

Generation: create new map chromosomes $\hat{m}_t^{(i)}$ based on inverse measurement model $p(m_t^{(i,j)} | x_t^{(i)}, z_t)$

Compress all new map chromosomes $m_t^{(i)} = compress(\hat{m}_t^{(i)}, \hat{m}_{t-1}^{(i)})$

}

The table 3-1 presents the notations used in this chapter and the values of major tuning parameters.

TABLE 3-1: MAJOR VARIABLE NOTATIONS FOR GAF MAPPING

Variable	Remarks
$(x_{0t}^{(i)}, m_t^{(i)})$	The i^{th} Rao-Blackwellised Particle
$x_t^{(i)}$	Latest robot pose
$m_t^{(i)}$	Chromosome map in the i^{th} Rao-Blackwellised Particle
$m_t^{(i,j)}$	The j^{th} map chromosome in the chromosome map $m_t^{(i)}$
z_t	Latest sensor measurements
$K_t^{(i)}$	The number of chromosomes for chromosome map in the i^{th} Rao-Blackwellised Particle
pm	Mutation rate: set at 0.002
pc	Crossover rate: set at 0.2
pr	Mutation range: set at 10% of maximum sensor range

The variables m , x , z , represent map, robot pose and sensor measurements respectively. Same notations are also used in other chapters.

3.4. Performance comparisons in 2D

For different map representations, their performances vary across different environments and different sensor configurations. Each has its advantages and disadvantages. Here, feature based maps, occupancy grid maps, compressed particle maps, and genetic algorithmic filter maps are discussed and compared under different assumptions. In this section, the range and bearing sensors (laser range finders/sonar) are considered.

3.4.1. The simulator

The simulator used in this research is developed based on Player/Stage projects [55] [56] and Orca projects [57]. The simulated environment is in 2D space. The 2D simulated environment is designed by drawing the bitmap images and setting the Player configuration files. During the simulation, the robot pose data and the sensor measurements are obtained from the Orca components and published through ICE-3.2.1. Our simulation component then receives the data. The data is used for SLAM or further processed with certain artificial noises/model-based approaches to simulate particular scenarios. Figure 3-6 demonstrates one example of a 2D simulated environment.

Figure 3-6: Example of 2D Player/Stage simulation environment.

As described on the websites [55], the Player project creates free software that enables research in robot and sensor systems. The Player robot server is probably the most widely used robot control interface in the world. Its simulation backends, Stage and Gazebo, are also very widely used.

- Player robot device interface

Player provides a network interface to a variety of robot and sensor hardware. Player supports a wide variety of mobile robots and accessories.

- Stage multiple robot simulator

Stage simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. The simulation environment is designed by drawing the bitmap and modifying the configuration files.

Besides the player/stage project, components of the ORCA project are also used in our simulations. Detailed descriptions of the Orca project can be found in [57]. The components used in our simulations are:

- Laser2d: Simulate laser data;
- Robot2d: Simulate robot motions and odometry data;
- Laser2Og, Ognode: Occupancy grid mapping.
- Localnav: to run the robot autonomously.

3.4.2. *Sensor models*

Although the Orca and Player projects only modelled limited types of sensors, there are several types of sensors used in our simulations and experiments, including laser range finders, sonar sensors and the stereo vision sensors. They are all bearing and range measurement devices with different noise models. The laser range finder shown in Figure 3-7 has the most accurate bearing and range sensors. The range estimates can be as accurate as 5 centimetres. Sonar sensors are less accurate and often difficult to model in a complicated environment. Here, sonar is also modelled

since it has unique advantages in an underwater environment. Stereo vision, although it has 3D sensors, can be used as a 2D bearing and range measurement device. The range measurement error of stereo vision is relatively small in the short range and large in the long range. Therefore, the actual maximum effective range of the stereo vision sensors is quite limited. However, they can be cheap and informative.

Figure 3-7: SICK 2D laser range finder

3.4.3. *2D Laser range finders*

Laser range finders:

For a simple 1D laser range finder, if the laser range finder gives a distance measurement, four types of basic assumptions can be made:

1. There is some object located at/near the distance measured (A)
2. There is no object located between the sensor and object (B).
3. Whether there are objects behind the object A is unknown (C).
4. The uncertainty of the range measurement is known or can be experimentally found out.

Figure 3-8: Simple 1D sensor model.

If the measurement is to be mapped into a 1D occupancy grid map, the weight calculated for the map update will be similar to Figure 3-8. Mathematically, the weight indicates the probability of the occupancy in that particular occupancy cell. In B, the weight is even less than that of an unknown space, indicating that there is ‘negative information’ obtained. On the other hand, the 1D model for a range-and-bearing sensor will often assume:

- The occupant object will generate sensor observations
- Empty space will not generate sensor observations.
- The laser/sonar beam cannot penetrate the objects.

The uncertainty of the range measurement is known or can be experimentally found out.

However, the above assumptions may not hold true. There are some objects which are transparent to the laser beams. In addition, the reflectivity of the objects also causes unreliable measurements. The sensor may not be able to receive the reflections from the objects with low reflectivity. Another well known problem is the so-called edge effect. When the laser beam hits the edge of the object, some of the beam will be reflected and other parts will be reflected by objects behind. As a result, the overall range measured will be unreliable on the edge of the object.

Between the edge of the object and the walls behind the object, there are some spurious measurements, indicating some ‘obstacles’ which actually do not exist.

In addition, the 2D laser range finder also contains bearing errors. The 2D laser range measurements are usually provided in a 1D array Z [num_of_laser_points] with a constant bearing interval. The accuracy of the angular measurement depends on the resolution of the laser beam's scan. Most of the 2D laser range finders on the market have angular resolution from $0.25 \sim 1$ degree. Furthermore, laser range finders suffer from biased angular errors. These errors might be caused by the mounting or the laser range finder itself. Therefore, the angular error has also been modelled to provide more accurate environmental mappings. The most common model is to model the angular error by a normal distribution pdf, as demonstrated in Figure 3-9 (generated by 10,000 samples with mean $\chi = 0$ and std $\sigma = 2$):

Figure 3-9: Normal distribution $N(0,2)$

If a biased angular error is considered, then the noise of angular measurement is modelled by its bias and standard deviation. Figure 3-10 gives an example of a biased distribution.

Figure 3-10: Biased distribution for the errors $N(2,2)$

3.4.4. *Feature based map*

In the simulations, the laser range finder is assumed to have a maximum range of 8.1 meters. The sensor model is the simple model described in Section 3.4.3. Both the measurement noises in range and bearing are assumed to be Gaussian. The standard deviation of the measurement noise in range is assumed to be 0.05 meters and the standard deviation of the bearing measurement is assumed to be 0.5° . The time interval for the processing is set to 0.1 second.

Since a feature based map extracts features (landmarks) from the environment, the advantage of the feature based map is that the sensor information can be reduced to control the computational cost. From [9] and [10] it can be observed that if the feature extraction functions well, the computational cost to process the features will be less than the computational cost required by view-based approaches.

To provide reliable feature extraction, the accuracy of the laser range finder is important. With the player-stage simulation environment and the feature extraction component developed in the ORCA project, polarfeatureextractor [57], the following simulations show the results.

Figure 3-11 show the feature extraction results for line features in a simulated indoor environment. The walls are extracted as line features marked by the double

line. A reliable line feature map can also be built.

Figure 3-11: Line feature extraction in simulation.

Left: Simulation environment with robot in red and obstacles in black. Right: line features extracted in double black line.

In the above two scenarios, the environment is assumed to contain line features. However, if the environment is unstructured, as shown in Figure 2-6, the above approaches would fail. The prior knowledge/assumptions used in Figure 3-9 will not hold true in the case of Figure 3-3. View-based approaches are significant complements to feature based approaches.

3.4.5. *Occupancy grid map*

The occupancy grid map is the most common view-based map representation. The memory consumption and computational cost are highly dependent on the grid resolution, i.e. the cell size ε . For the 2D grid map as shown in Figure 3-9, the laser range finder scans are mapped into grid maps. The black cells represent the obstacles, grey areas represent the unknown space and the white cells represent free space.

Occupancy grid maps have been widely used and verified by many researchers and implementations. The results are as shown in Figure 3-12. The occupancy grid map represents the environment to the setting accuracy.

Figure 3-12: Occupancy grid mapping result for a static robot.

Blue: robot and laser scans. Black: obstacles. White: empty space. Grey: unknown space.

When the robot is moving, the whole simulation environment can be mapped. The laser readings and robot pose information are obtained from the simulator component. The occupancy grid map can be used to map the environment as shown in Figure 3-13.

Figure 3-13: Occupancy grid map. Blue: robot, target and laser scan. Black: obstacles. White: empty space. Grey: unknown space.

As discussed in [13], the total number of grid cells is denoted as M , the number of cells which are swept and updated as obstacles is denoted as A , here the number of cells that were updated as free space is denoted F . The computational cost of updating one fine occupancy grid map is proportional to $A + F$. The memory consumption is proportional to M .

3.4.6. Compressed particle filter for map representation

If the environmental map is represented by standard particles, for each interval, the particles are generated according to the sensor measurements. The number of particles increases proportionally to the time even though the robot is stationary. This will result in an unbounded number of particles and require unlimited computational resources.

By applying the compressed particle approach (with a resolution of size ε), the environmental map can be represented by compressed particles without an unnecessary increase in the number of particles. As shown in Figure 3-14, the number of compressed particles is bounded so that the computational cost will be independent of the time and only proportional to the number and size of the obstacles.

Figure 3-14: Number of particles for a particle map

If the robot is stationary, the number of compressed particles will not exceed an upper bound. However, the uncertainty of the sensor measurement caused a large number of compressed particles. The efficiency can be improved by the resampling process. Figure 3-12 shows the resampling results over a compressed particle map with the Monte Carlo resampling technique.

The same data sets are also processed by the Monte Carlo particle mapping. In the initialization stage, the particle map is as shown in Figure 3-15. After 20 Monte Carlo resamplings [58], particles are discarded and the resultant map representation is not acceptable.

Figure 3-15: Monte Carlo resampling results.

(a) particle map before resampling; (b) particle map after 20 resamplings.

It can be observed that for some areas, especially on the curved/line obstacle edges, the compressed particle representation becomes discarded. The resultant map representation is not acceptable. The Figure 3-16 shows one example of the count values histogram for the compressed particles after 20 iterations. After a number of Monte Carlo re-samplings, it can be observed that a few compressed particles dominate the map representation. Such a map representation is not acceptable

Figure 3-16: Histogram for the count values of compressed particle map

The adaptive resampling technique can mitigate this shortcoming. However, when

the robot is moving, it is difficult to determine when the resampling should or should not start. In [10], the diversity of the particles representing the robot pose is calculated. Only when the diversity exceeds a threshold does the resampling take place. However, the diversity of the compressed particles is large and varies greatly in different areas of the environment. Therefore, the adaptive resampling technique is difficult to apply to the map representation.

Artificial noise [20] can also mitigate this shortcoming. However, for the map representation, new particles are constantly generated due to the sensor measurement. Such an approach didn't mitigate the problem because the compressed particles with high count values will be very 'strong' in Monte Carlo resampling. The stability of the particle map in the resampling process is, due to the sample impoverishment problem, not assured.

3.4.7. *Genetic algorithmic filter mapping*

Due to these limitations, genetic algorithm techniques are used to improve the compressed particle filters, resulting in the genetic algorithmic filter (GAF) for environmental map updates. The same simulations are tested as in Figure 3-14 with the genetic algorithmic filter. The robot platform is assumed to be static at the starting position. The measurements from the simulated 2D laser range finder are collected. The data is then processed and mapped by the GAF approach to construct a non-parametric chromosome map. The created chromosome map is expected to have three basic properties:

- **Efficiency:** The total number of chromosomes is small. It is proportional to the objects observed, but not to the environment space size or the navigation time elapsed.
- **Approximation:** The chromosome map must approximate the environment map.
- **Stability:** The chromosome map is stable and will not be discarded after several resamplings.

The same data sets are then processed by the GAF mapping module for comparison. The GAF can maintain the map chromosomes, avoiding the problems of Monte Carlo resampling shown in Figure 3-15. The final chromosome map is shown in Figure 3-17.

Figure 3-17: Simulation results of GAF mapping. Red: indicated robot trajectory. Black: obstacles represented by map chromosomes. White: empty space.

It can be observed that the genetic algorithmic filter will not cause the particles to be discarded. The map representation is reliable and stable after more than 20 resampling processes. Since the genetic algorithmic techniques are used, the compressed particles used to represent the environmental map are now called map chromosomes. The number of chromosomes can also be bounded after few time intervals and the map resolution can be determined by the virtual chromosome size ε . The overall number of chromosomes is less than the number in the

compressed particle map, showing an improvement in memory efficiency.

When the robot is moving, assuming the robot pose/trajectories are known, the environmental map can be then be built by mapping the sensor measurement with the genetic algorithmic filter approach. The resulting chromosome map is shown in Figure 3-17, with the same 2D data used in Figure 3-8.

In Figure 3-18, the number of the chromosomes in the map is shown with respect to the number of time intervals (sec). It can be observed that the number of map chromosomes increases when the robot is moving and detects new obstacles. However, when the robot is travelling in an ‘observed’ area (time 100 to 200), the number of map chromosomes will not increase because the robot is travelling into the observed environment.

Figure 3-18: The number of chromosomes for single GAF map

Comparisons between occupancy grids, the Monte Carlo particle filters map, and the GAF map show that both the occupancy grid map and the GAF map can represent the environment accurately. However, the computational costs of these

two approaches are different. The computational cost is proportional to the number of cells and the number of map chromosomes. In Figure 3-19, the number of cells and the number of map chromosomes are shown with respect to different cell sizes for the 16 m by 16 m map. It can be observed that with high resolution the number of map chromosomes is significantly less than the number of grid cells. This indicates that the GAF map is more computationally efficient than the grid map.

Figure 3-19: Number of map chromosomes in GAF vs. number of grid cells

3.5. 3D GAF mapping results

The Velodyne laser range finder, as shown in Figure 3-20, was developed in 2005 by Velodyne LiDAR, Inc. In the DARPA challenge in 2005, Team Digital Auto Drive (DAD) competed in the second DARPA Grand Challenge. After that, more and more robotics teams developed their own Velodyne-mounted vehicles, especially in the DARPA Urban Challenge. As described by the manufacturer, the Velodyne is a 64-element LiDAR sensor that delivers a 360-degree HFOV (horizontal field of view) and a 26.8-degree VFOV (vertical field of view). It features frame rates of 5-15 Hz and over 2.2 million data points per second with range errors within 0.2m. The point cloud it generates is a massive leap forward in terrain and environment perception. Both distance and intensity data can be obtained through the Ethernet output packet payload.

The Velodyne was used by five out of six of the finishing teams at the 2007 DARPA Urban Challenge, including the winning and second place teams. Some

teams relied exclusively on the Velodyne for navigation. The final performance strongly proves that Velodyne is one of the best laser range finders today for autonomous vehicles. Nowadays, Velodyne has become more and more popular in the robotics community. It is also in use for ground truth testing, military autonomous navigation, and other mapping / surveying applications.

The process and the modelling of the 3D laser range finder are more complicated than the 2D laser range finder. The details are introduced in the Appendix A.

Figure 3-20: Velodyne 3D laser range finder

The experiments in this chapter were conducted in the NTU campus. Figure 3-21 shows the experimental environment (NTU campus map) for Velodyne 3D LiDAR.

Figure 3-21: NTU campus map: experiment environment for Velodyne 3D LiDAR.

In the 3D environment, the computational cost for mapping the obstacles increased significantly. To achieve efficient and reliable mapping results, a GAF mapping should fulfill the following requirements:

- When the robot is stationary, the number of map chromosomes should be stable.
- The map chromosomes should closely approximate the 3D environment and the obstacle surfaces.
- In the resampling process, the chromosome map should be stable.

To verify the functionality of the 3D GAF-mapping, the test was conducted in NTU and the number of map chromosomes for a single map is recorded. The robot is static and the environment is also static. The number of map chromosomes in a single map is recorded. Due to the GAF resampling, the number of map chromosomes varies at different times. However, as shown in Figure 3-22, the

number of map chromosomes will not increase indefinitely. The mean number of the number of map chromosomes is 6350 and the standard deviation is 563.

Figure 3-22: Number of map chromosomes for static 3D mapping

The GAF mapping result is shown in Figure 3-23 in a 45 degree view. The black/grey dots represent the map chromosomes. The darkness represents the height of these map chromosomes. Grey indicates lower obstacles (e.g. ground) and the dark indicates the higher obstacles (e.g. buildings and walls). The red dot indicates the position of the robot. The mapping result is shown together with the campus view in Figure 3-24 and Figure 3-25. The locations A and B marked in 3D GAF mapping result (Figure 3-21) are pointed out in the campus view in Figure 3-24 and Figure 3-25, respectively. It can be seen that the map chromosomes nicely approximate the environment after more than 40 GAF re-samplings. The sample impoverishment problem related to the Monte Carlo resampling can be mitigated by 3D GAF mapping as well.

Figure 3-23: 3D GAF mapping result with location A and B marked (45 degree view)

Red: robot trajectory. Black: map chromosomes.

Figure 3-24: 3D environment in campus view with location A marked

Figure 3-25: 3D environment in campus view with location B marked

Figure 3-26: Details of 3D GAF mapping results. (45 degree view)

Red: robot trajectory. Black: map chromosomes.

In Figure 3-26, we zoom the 3D GAF mapping result to present the details. It can be observed from the mapping results that typical urban obstacles like lamp post, walls, and curbs can be mapped. The moving car nearby is also detected but it will vanish after a short time, similar to what will happen on an occupancy grid map.

Compared with occupancy grid mapping, the GAF mapping improves in efficiency by the resampling process. Resampling mitigates the sample depletion problem and reduces the number of map chromosomes. Such advantages can be observed in 3D mapping as well. In Figures 3-28 and 3-29, the 3D GAF mapping results with different resolutions (virtual chromosome size) are shown, together with the number of chromosomes shown in Figure 3-27 for various virtual chromosome sizes. In the 3D environment, the number of grid cells will increase as the cube of the environment size. In Figure 3-30, the relative number of grid cells and the relative number of map chromosomes is shown with the number at size 60 cm being set to 100. It can be observed that, with higher resolution, the number of chromosomes will increase but not as the cube. For example, with a virtual chromosome size of 0.1 m, the number of chromosomes is less than eight times that with a virtual chromosomes size of 0.2 m. Due to the resampling process, the number of map chromosomes may vary at different times but, overall, the number of map chromosomes is stable and efficient.

Figure 3-27: Number of chromosomes for various virtual chromosome sizes

Figure 3-28: 3D mapping result with virtual chromosome size 60 cm (45 degree view)

Red: robot trajectory. Black: map chromosomes.

Figure 3-29: 3D mapping result with virtual chromosome size 10 cm (45 degree view)

Red: robot trajectory. Black: map chromosomes.

Figure 3-30: Relative computational cost (the cost at size 60 cm is set at 100)

This advantage demonstrates that 3D GAF mapping is more efficient than occupancy grid mapping, especially at higher resolution.

Because the blue-print of the building is not available, it is difficult to find reliable data as ground truth. Here, to verify the accuracy of the mapping results, manual measurements were taken. The results are presented in table 3-2.

TABLE 3-2: COMPARE MAPPING RESULT WITH GROUND TRUTH

	Manual Measurement	Map result
Road width 1	8.02	8.04
Road width 2	7.95	8.06
Road width 2	8.10	7.97
Gate width	7.05	7.13

The widths of the road are measured manually three times as well as the width of the car park gate. The manual measurements are compared with the values obtained from the chromosome map. The mapping results are very close to the measured ground truth. The error is less than 15cm.

3.6. Summary

This chapter presented and discussed the metric based map representation approaches which are commonly used in simultaneous localization and mapping.

Although the feature based map is efficient and effective in certain environments with known prior knowledge, mobile robot navigation in complex, unstructured, unknown environments requires view-based map representation approaches. Occupancy grid maps do not rely on features but are not efficient. Particle maps suffer from the sample impoverishment problem and so are not suitable for mapping.

The genetic algorithmic filter approach presented is combined with the compressed particle filter and the genetic algorithm technique. For both 2D and 3D environment, the sample impoverishment problem of the particle maps can then be mitigated. When the environmental map is represented by chromosomes and updated by the genetic algorithmic filter, its performance shows advantages over the occupancy grid map. With the same desired resolution, the GAF approach is more efficient in memory consumption, especially for 3D space. The resampling process makes the GAF approach more computationally efficient in map representation.

CHAPTER IV: RAO–BLACKWELLIZED GENETIC ALGORITHMIC FILTER SLAM

4.1. Introduction

The practical objective of simultaneous localization and mapping (SLAM) algorithms is to estimate the joint posterior of robot pose and environmental map consistently and efficiently. Murphy, Doucette and colleagues introduced the *Rao–Blackwellized* particle filter [1]. In this approach particles are used to represent the posterior over some variables, together with other parametric posterior density functions to represent all other variables. This *Rao–Blackwellized* particle filter approach provides for the factorization of the SLAM problem so that the full robot posterior can be recovered. This mathematical framework has since been extended and implemented in robot navigation by many researchers.

In the last chapter, the compressed particle map representation was proposed for the Rao–Blackwellized particle filter SLAM framework to recover the full robot posterior. Environmental maps are represented by groups of compressed particles. Compressed particle maps enable the full use of sensor accuracy without increasing memory consumption; an arbitrary measurement model can be implemented based on mathematically sound posterior estimation algorithms. In addition, negative sensor information can still be recovered from the recorded data for identifying moving objects or spurious noises.

Due to the limitations of particle resampling, it is necessary to design adaptive techniques to reduce the real-time computational costs without loss of performance to be practically feasible. Apart from sample size, sample impoverishment and particle depletion are major issues with particle filter based approaches. Heuristic methods are introduced to mitigate these problems. Genetic algorithms (GA) have also proved to be effective in this regard [59] [60]. Similarities between GA and particle filters have been pointed out long ago. Techniques in GA have been applied to mitigate problems in robot navigation with particle filters. The resultant genetic

algorithmic filter (GAF) improved the robustness of the compressed particle map. Here, to avoid confusion with the Rao–Blackwellized particles, the compressed particle map improved by the GAF is also called the chromosome map. A chromosome is the element in genetic algorithms, and consists of an array of genetic code. Similarly, the element of the chromosome (particle) map, or say, the map particle in the compressed particle map, is called a map chromosome. Combining the chromosome map representation with the Rao–Blackwellized particle filter SLAM framework, yields the proposed Rao–Blackwellized Genetic Algorithmic Filter SLAM (RBGAF-SLAM). This chapter will describe the details of RBGAF-SLAM as well as a variant.

The rest of this chapter is structured as follows: Section 4.2 briefly describes mathematical formulations related to particle filters and Rao–Blackwellized particle filter SLAM. Section 4.3 presents our novel RBGAF-SLAM algorithm. Section 4.4 gives a detailed discussion of the computational complexity and memory consumption of DP-SLAM and RBGAF-SLAM. The simulation and experimental results are shown in Section 4.5 to demonstrate the effectiveness and efficiency of RBGAF-SLAM. Section 4.6 presents improvements for RBGAF-SLAM in dynamic environment and genetic learning. Section 4.7 gives a summary.

4.2. Basic formulations

4.2.1. Particle Filters

In particle filters, the posterior distribution is represented using samples known as particles and is denoted as $X_t := x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[N]}$. Each particle $x_t^{[i]}$ ($1 \leq i \leq N$) is a concrete instantiation of the state (robot pose) at time t , a hypothesis as to what the true state may be at time t . N denotes the number of particles in the particle set X_t . The intuition behind particle filters is to approximate the belief $bel(x_t)$ by the set of particles X_t . The likelihood for a state hypothesis x_t to be included in the particle set X_t is proportional to its Bayesian filter posterior $bel(x_t)$. The input of this algorithm is the particle set X_{t-1} , along with the most recent control u_t and the

most recent measurement z_t . For map-aided localization, if an environment map m_t is always available, the robot pose's posterior at time t is thus obtained as

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m_t) \overline{bel}(x_{t-1}) dx$$

$$bel(x_t) = \eta p(z_t | x_t, m_t) \overline{bel}(x_t)$$

The particle filter algorithm can be represented as:

At time t , initialize particle sets $\bar{X}_t = X_t = \emptyset$.

For all particles, $i = 1, \dots, N$.

{

Sample: $x_t^{[i]} \sim p(x_t | u_t, x_{t-1}^{[i]})$,

Calculate weights: $w_t^{[i]} = p(z_t | x_t^{[i]})$

$\bar{X}_t = \bar{X}_t + \{x_t^{[i]}, w_t^{[i]}\}$.

}

Resampling select $x_t^{[i]} \propto w_t^{[i]}, i = 1, \dots, N$.

Add $x_t^{[i]}$ into X_t .

More detailed information on particle filters for mobile robot navigation can be found in [12].

4.2.2. Basic notations for RBGAF-SLAM

In 2D Cartesian coordinates as in Figure 3.2, the initial position is assumed to be the origin of the coordinate frame and the initial heading direction is assumed to be 0, i.e., the robot is facing the $\vec{a}$ axis at the starting position. Then the robot pose

(position and orientation) at time instance t will be $x_t = ({}^r a, {}^r b, {}^r \theta)$. The robot trajectory will be denoted as $x_{0:t} = \{x_0, x_1, x_2, \dots, x_t\}$. If there are multiple robot trajectories with ID $i = 1, 2, \dots, P$, the trajectories are $x_{0:t}^{(i)} = \{x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, \dots, x_t^{(i)}\}, i = 1, 2, \dots, P$.

Besides the coordinate notation, the SLAM problem is best described as a probabilistic Markov chain. The motion model and the sensor model are fundamental models of probabilistic SLAM approaches.

Poses evolve according to a probabilistic law, often referred to as the motion model based on control $u_{0:t-1}$ and the robot trajectory $x_{0:t-1}$:

$$p(x_t | u_{0:t}, x_{1:t-1})$$

The motion model is usually a time invariant probabilistic generalization of the robot kinematics. The models vary with different robot platforms, sensors and environments. One of the advantages for RBGAF-SLAM is that it does not have any restrictions or assumption for these models so that arbitrary models can be used with the RBGAF-SLAM framework. The models provided in this thesis are just examples for the convenience of presenting RBGAF-SLAM. The example of a sensor model has been presented in Section 3.3.

4.2.3. Rao–Blackwellized particle filter SLAM

Rao–Blackwellized particle filters are an effective means to solve the full posterior of the robot pose. Each Rao–Blackwellized particle of $\{(x_{0:t}^{(i)}, m_t^{(i)}), i = 1, \dots, P\}$ consists of one robot trajectory $x_{0:t}^{(i)}$ and one associated environmental map $m_t^{(i)}$. The key idea of the approach is to solve the recursive Bayes filter update by the equations

$$p(x_{1:t}, m | z_{1:t}, u_{0:t-1}) =$$

$$p(m | x_{1:t}, z_{1:t}, u_{0:t-1}) p(x_{1:t} | z_{1:t}, u_{0:t-1}).$$

The whole Rao–Blackwellized particle sets $\{(x_{1:t}^{(i)}, m_t^{(i)}), i=1, \dots, P\}$ are resampled based on the likelihoods of the measurements $z_{1:t}$ in the maximum likelihood map constructed by the corresponding robot trajectory. In [13], naïve SLAM based on the Rao–Blackwellized particle filter is introduced. Each particle consists of one robot trajectory and an associated environmental map in the form of an occupancy grid. It can be shown as follows:

Take robot trajectory, sensor measurements and control inputs $x_{1:t-1}, z_{1:t}, u_{0:t}$

Predict new robot pose and update robot trajectory based on proposal distribution π :

$$\hat{x}_t^{(i)} \sim \pi(x_t | z_{1:t-1}, u_{0:t})$$

$$\hat{x}_{0:t}^{(i)} \sqsupset (\hat{x}_t^{(i)}, x_{0:t-1}^{(i)})$$

The importance weight $w_t^{(i)}$ is then calculated as:

$$w_t^{(i)} \propto \frac{p(x_t^{(i)} | z_{1:t}, u_{0:t})}{\pi(x_t^{(i)} | z_{1:t-1}, u_{0:t})}$$

Resample the particle (trajectory/map pair) according to the importance weight $w_t^{(i)}$

$$x_{0:t}^{(i)} \sim w_t^{(i)}$$

Update the occupancy grid map based on observation and new robot pose

$$m_t^{(i)} \sim p(m_t | x_{0:t}^{(i)}, z_{1:t})$$

Repeat for the next interval. In this thesis, the tilde symbol, $\sim$, denotes the sampling (or selection) operations.

4.3. Description of the RBGAF-SLAM

To achieve an efficient Rao–Blackwellized SLAM and overcome its shortcomings, we proposed the Rao–Blackwellized Genetic Algorithmic Filter SLAM. The key

idea is to incorporate and integrate genetic algorithmic techniques into RBPF so that environment maps are represented by sets of chromosomes (i.i.d particles).

In RBGAF-SLAM, each particle of $\{(x_{0:t}^{(i)}, m_t^{(i)}), i = 1, \dots, P\}$ consists of one robot trajectory $x_{1:t}^{(i)}$ and one associated environmental map $m_t^{(i)}$. The environmental map $m_t^{(i)}$ is represented by a number of chromosomes, separated into island groups. The j th chromosome in the i th particle (trajectory/map pair) at time t is then denoted as $m_t^{(i,j)}$. Altogether, there are P robot trajectories and P chromosome maps, with the index i from 1 to P .

4.3.1. Algorithmic description of the RBGAF-SLAM

The basic procedures of RBGAF-SLAM are similar to the grid RBGAF-SLAM (naïve SLAM). However, RBGAF-SLAM has unique map representation and mapping process as introduced in Chapter III. The genetic algorithmic filter can also be used in the particle (trajectory/map pair) resampling process.

Let the robot start at time $T = 0$, then at time $T = t - 1$, the i_{th} Rao-Blackwellized Particle (trajectory map pair) will be $(x_{0:t-1}^{(i)}, m_{t-1}^{(i)})$. At the time instance $T = t$, the new odometry (control) data obtained is u_t and the new laser range finder measurement (or other range-bearing sensor) is z_t . Based on a motion estimation model, the new hypothesis of the robot trajectories will be generated as the proposal distribution π for the Rao-Blackwellized genetic algorithmic filter. Here, $\bar{x}_t$ is the hypothesis of the proposal distribution for the estimated robot pose.

$$\bar{x}_t^{(i)} \sim \pi(x_t | x_{0:t-1}^{(i)}, z_{1:t-1}, u_{0:t}), i = 1, 2, \dots, P$$

Then calculate the importance weight of the Rao-Blackwellized particle (robot trajectory and the associated map) according to the latest sensor measurement:

$$w_t^{(i)} \propto \frac{p(x_t^{(i)} | z_{1:t}, u_{0:t})}{\pi(x_t^{(i)} | z_{1:t-1}, u_{0:t})}$$

The final updated Rao–Blackwellized particle will be generated based on the importance weight $w_t^{(i)}$ through GAF.

$$x_{0:t}^{(i)} \stackrel{GAF}{\sim} w_t^{(i)}, i = 1, 2, \dots, P$$

Then update the map according to the new sensor measurement (see section 3.4.3) and the estimated robot pose using GAF mapping

$$m_t^{(i)} \stackrel{GAF}{\sim} p(m_t | x_t^{(i)}, m_{t-1}^{(i)}, z_t), i = 1, 2, \dots, P$$

Repeat such update for the next time interval.

In RBGAF-SLAM, the Rao-Blackwellised particle is the combination of robot trajectory and associated chromosome map. To avoid confusions in notations, the table 4-1 lists major notations for RBGAF-SLAM used in this chapter.

TABLE 4-1: VARIABLE NOTATIONS

Variable	Remarks
$(x_{0:t}^{(i)}, m_t^{(i)})$	The i^{th} Rao-Blackwellised Particle
$x_{0:t}^{(i)}$	Robot trajectory in the i^{th} Rao-Blackwellised Particle
$x_t^{(i)}$	Latest robot pose
$\bar{x}_t^{(i)}$	Temporary estimated robot pose
$m_t^{(i)}$	Chromosome map in the i^{th} Rao-Blackwellised Particle
$m_t^{(i,j)}$	The j^{th} map chromosome in the chromosome map $m_t^{(i)}$
$z_{0:t}$	Sensor measurements
z_t	Latest sensor measurements
$u_{0:t}$	Control outputs or odometry readings

The figure 4-1 illustrates the procedures of RBGAF-SLAM. In every time interval, both robot trajectories and associated chromosome maps are updated with GAF.

Figure 4-1: Illustrated RBGAF-SLAM procedures

RBGAF-SLAM can implement an arbitrary model or sensor measurement model theoretically. Here, in the following descriptions of this section, we demonstrate one example of a RBGAF-SLAM based on common approaches used in motion estimation and sensor modeling. The mobile robot navigation is assumed to take place in a 2D plane and altitude information is ignored in the formulation. However, it can be easily extended to 3D RBGAF-SLAM.

4.3.2. Example of motion model

The motion model can range from very simple to very complicated. In our simulations and experiments, the motion model is based on wheel encoders and a gyroscope. The wheel encoders provide the velocity information and the gyroscope provides the heading information. However, other motion models can be implemented in RBGAF-SLAM. There is no restriction on the choice of motion models.

The robot is assumed to move in a 2D plane. The encoder can record the angular movement of two wheels (ω_L, ω_R) and the gyroscope provides the bearing angle reading $\hat{\theta}$ to the initial position, CCW. The error of the wheel encoder is assumed to be proportional to the distance travelled Δd and the error of the gyroscope is assumed to be Gaussian.

Figure 4-2: A two-wheel robot with width L and wheel radius r

The robot pose in the Cartesian coordinates is the position (a, b) and the bearing angle θ , CCW. The motion estimation model is linear and separated into small motion pieces for each time interval Δt . Let the estimated robot pose at time $T=t-1$ be $x_{t-1} = (a_{t-1}, b_{t-1}, \theta_{t-1})$ and the real robot pose, $\bar{x}_{t-1} = (\bar{a}_{t-1}, \bar{b}_{t-1}, \bar{\theta}_{t-1})$. At time $T=t$, the real angular velocities of the two wheels are $(\bar{\omega}_L, \bar{\omega}_R)$ and the real heading is $\bar{\theta}$.

Therefore, as shown in Figure 4-2 and Figure 4-3 (the trajectory of the robot is marked in red) with the wheel diameter r and the time for one interval Δt , the displacement is approximated by the distance travelled Δd , which is estimated as

$$\Delta d = \frac{(\omega_L + \omega_R) \cdot r \cdot \Delta t}{2}$$

The angle θ' is approximated as the average value of the robot headings

$$\theta' = \frac{\theta_{t-1} + \hat{\theta}}{2}.$$

Figure 4-3: Simple motion model for two-wheel mobile robot

The new robot position can be estimated as:

$$a_t = a_{t-1} + \Delta d \cdot \cos(\theta')$$

$$b_t = b_{t-1} + \Delta d \cdot \sin(\theta')$$

However, there are the noises in both distance and bearing measurement. The mean and variance of distance measurement is denoted as η_d, σ_d , respectively. The mean and variance of bearing measurement is denoted as $\eta_\theta, \sigma_\theta$ respectively. Therefore, if at time $T = t - 1$, the robot pose is $x_{t-1} = \{a_{t-1}, b_{t-1}, \theta_{t-1}\}$ and at time $T = t$ the odometry reading is $u_t = \{(\omega_L, \omega_R), \theta_t\}$, the new robot pose will be estimated to be $x_t = \{a_t, b_t, \theta_t\}$, in which:

$$a_t = a_{t-1} + (\Delta d + N(\eta_d, \sigma_d)) \cdot \cos(\theta' + N(\eta_\theta, \sigma_\theta))$$

$$b_t = b_{t-1} + (\Delta d + N(\eta_d, \sigma_d)) \cdot \sin(\theta' + N(\eta_\theta, \sigma_\theta))$$

$$\theta_t = \hat{\theta} + N(\eta_\theta, \sigma_\theta)$$

This estimation is less accurate but it is computationally efficient. It is used in the simulations and experiments for all the robots, including the four-wheel vehicles in this thesis.

4.3.3. Robot pose estimation in RBGAF-SLAM

For each Rao–Blackwellized particle, an array of robot poses is maintained to represent a hypothesis as to the actual robot trajectory. The array contains the robot poses from $T=0$ to $T=t$ as one hypothesis of the robot trajectory. Let $i=0,1,2,\dots,P$. If there are P Rao–Blackwellized particles, the element to represent a robot pose at time t of the trajectory array in the i^{th} Rao–Blackwellized particle is $x_t^i = \{a_t^i, b_t^i, \theta_t^i\}$, in which a , b , and θ represent a hypothesis of the robot pose.

The representation of the robot trajectory is then

$$x_{0:t}^{(i)} = \{x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_{t-1}^{(i)}, x_t^{(i)}\}$$

All the elements at the time $T = t$ approximate the pdf of the robot pose at the time instance t .

$$x_t^{(i)} = \{a_t^{(i)}, b_t^{(i)}, \theta_t^{(i)}\}, i = 1, 2, \dots, P$$

At the initial position, the robot pose is $(0, 0, 0)$ at the time instance $T=0$.

$$x_0^{(i)} = \{a_0^{(i)} = 0, b_0^{(i)} = 0, \theta_0^{(i)} = 0\}, i = 1, 2, \dots, P$$

At time $T = t - 1$, with the odometry sensor readings $u_t = \{\omega_L, \omega_R, \hat{\theta}_t\}$, the robot pose is estimated as the motion model (proposal distribution) presented in Section 4.3.2:

$$a_t = a_{t-1} + (\Delta d + N(\eta_d, \sigma_d)) \cdot \cos(\theta' + N(\eta_\theta, \sigma_\theta))$$

$$b_t = b_{t-1} + (\Delta d + N(\eta_d, \sigma_d)) \cdot \sin(\theta' + N(\eta_\theta, \sigma_\theta))$$

$$\theta_t = \hat{\theta} + N(\eta_\theta, \sigma_\theta)$$

New poses are added into the robot trajectory for all the particles before the resampling process.

$$x_{0:t}^{(i)} = \{a_{0:t}^{(i)}, b_{0:t}^{(i)}, \theta_{0:t}^{(i)}\}, i = 1, 2, \dots, P$$

Here, $a_t^{(i)}, b_t^{(i)}, \theta_t^{(i)}$ are three floating values representing the location of the robot pose at time t in 2D space. An importance weight value will be calculated for the final resampling of the trajectory/map pair.

4.3.4. Particle (trajectory/map pair) evaluation

The Rao–Blackwellized particles (trajectory/map pairs) are evaluated by the likelihood between the robot trajectory and the associated particle map. Similar to [1], the importance weight will be:

$$w_t^{(i)} \propto \frac{p(x_t^{(i)} | z_{1:t}, u_{0:t})}{\pi(x_t^{(i)} | z_{1:t-1}, u_{0:t})} = \int p(z_t | m_t^{(i)}, x_{0:t}^{(i)}) p(m_t^{(i)}) dm_t^{(i)}$$

However, such integration is quite complicated. If we represent the probability in logarithms, the multiplication operations will become sums. Since the environmental maps are represented by chromosomes $m_t^{(i,j)}$, the integration can be approximated as:

$$w_t^{(i)} \approx \sum_{j=1}^{K_t^{(i)}} (L(z_t | m_t^{(i,j)}, x_{0:t}^{(i)}) + L(m_t^{(i,j)})),$$

in which the function $L(\cdot)$ indicates the logarithm of the probability. The final importance weight is also, then, in logarithms.

To calculate the importance weight, all the previous robot poses are taken into account, along with the sensor measurement and the most current associated particle map. In logarithmic form it can be calculated as:

$$L(z_t | m_t^{(i,j)}, x_{0:t}^{(i)}) \approx \sum_{T=0}^t (L(x_T^{(i)} | m_t^{(i,j)}, z_T) + L(z_T))$$

Since the weight of the sensor measurement model is already in the logarithmic format, the function L can be obtained directly from the simple sensor model for ease of the implementation. However, the sophisticated sensor model can also be implemented to obtain a more accurate result. At any given time $T = t$, for the sensor measurement $z_t = \bigcup_{n=0}^{360} z_t^{(n)}$, each element represents a single laser beam measurement. The particular laser range measurement obtained from the laser beam should be ‘reasonable’ from the point of view of the robot pose at the time. The environment is represented by the particle map

$$m_T^{(i,j)} = ({}^m a_T^{(i,j)}, {}^m b_T^{(i,j)}, {}^m Count_T^{(i,j)}), j = 1, 2, \dots, K.$$

The relative position of the map particles are calculated with respect to the robot pose at that particular time $T = t$.

$$l_T^{(i,j)} = \sqrt{{}^m a_T^{(i,j)} * {}^m a_T^{(i,j)} + {}^m b_T^{(i,j)} * {}^m b_T^{(i,j)}}$$

$$h_T^{(i,j)} = \arctan({}^m b_T^{(i,j)}, {}^m a_T^{(i,j)})$$

With respect to the sensor measurement of range and bearing obtained $z_T^{(n)} = (r_T^{(n)}, \alpha_T^{(n)}), n = 0, 1, 2, \dots, 360$, if the sensor model denotes to $M(\cdot)$, then it will be

$$L(x_T^{(i)} | m_t^{(i,j)}, z_T) = \sum_{n=0}^{360} M(r_T^{(n)}, \alpha_T^{(n)}, l_t^{(i,j)}, h_t^{(i,j)})$$

Because it is unlikely for one map chromosome to be observed by all the laser beams (usually only one beam), practically, the calculation can be simplified as:

$$L(x_T^{(i)} | m_t^{(i,j)}, z_T) = M(r_T^{(n)}, \alpha_T^{(n)}, l_t^{(i,j)}, h_t^{(i,j)})$$

The overall importance weight for the Rao–Blackwellized particle (trajectory/map pair) is then:

$$w_t^{(i)} \approx \sum_{j=1}^{K_t^{(i)}} \left(\sum_{T=0}^t (M(r_T^{(n)}, \alpha_T^{(n)}, l_t^{(i,j)}, h_t^{(i,j)}) + L(z_T)) + L(m_t^{(i,j)}) \right)$$

This importance weight will be the evaluation result for the resampling stage.

4.3.5. *Resampling*

After the weights are calculated according to the total posterior for each particle $(x_{0:t}^{(i)}, m_t^{(i)})$, the complete collection of pairs is resampled. GA techniques are also applied to the particles. The techniques include crossover, mutation, and tournament selection as described in Section 3.4.

Although copying the robot trajectory and chromosome map is costly, the computational resources required to do this process are much less than copying the occupancy grid maps. Therefore, unlike DP-SLAM, if we associate robot trajectories to the map chromosomes, the computational cost will not be reduced. This is because that the chromosome map is significantly more efficient in memory consumption compared to the occupancy grid map at the same resolution. This computational cost is not the dominant term for the RBGAF-SLAM and it is acceptable for current processors. Section 4.4 discusses the computational costs in more detail.

4.3.6. *Map update*

The map evaluation and update process is completed by a GAF mapping as presented in Chapter III. The map chromosomes $m_t^{(i)}$ are evaluated and updated according to the robot trajectory $x_{0:t}^{(i)}$, the sensor measurement z_t , and the existing map chromosomes at the previous interval $m_{t-1}^{(i)}$. This can be denoted as:

$$m_t^{(i)} = \text{MapUpdate}(m_{t-1}^{(i)}, x_{0:t}^{(i)}, z_t)$$

The update process consists of map chromosome generation, compression, evaluation and selection, as shown in Chapter III, Section 3.3.

4.4. Computational complexity

If the map is an occupancy grid of size M and P particles are maintained, ignoring the cost of localization, $O(MP)$ operations must be performed merely copying maps. Practically, this approach is naïve in that computational resources are not sufficient to complete this task in most cases. Suppose the laser sweeps out an area A of intersecting occupancy grids, it can be found out that $A \ll M$. Therefore, DP-SLAM gives a more efficient way of solving this problem.

Instead of associating maps with particles, DP-SLAM associates particles with maps. It maintains just a single occupancy grid; each grid square stores a balanced tree. The tree is keyed on the ids of the particles that have made changes to the occupancy of the square.

Letting D be the depth of the ancestry tree for robot trajectories, the computational cost of DP-SLAM can be estimated as:

- (ADPlgP) operations for localization.
- (APlgP) operations to insert new data into the tree.
- Ancestry tree maintenance with amortized cost $O(D \cdot \text{ADPlgP})$

Because in most environments $M \gg A$ and $M \gg D \cdot \text{APlgP}$, DP-SLAM will be more efficient than Naïve SLAM.

In RBGAF-SLAM, P particles are maintained. For each particle, a group of chromosomes are associated with one robot trajectory. To represent the sensor measurement, n chromosomes are required. The number of groups is small, having at most 2 to 3 layers. The computational cost of searching a particular group is negligible. Then, the following computational complexity can be estimated.

For localization, each particle needs to make $O(nDP \lg P)$ accesses to the map. Furthermore, the cost of copying environment maps during the resampling stage can also be estimated as $O(nP)$. The cost of updating the robot trajectory is at most $O(D \lg P)$, and can be considered as $O(\lg P) \ll O(nP)$ in the usual case. To resample the environment map, $O(nP)$ tournament selections are needed.

In DP-SLAM, when initializing just a single grid map, the memory consumption is at least $O(M)$. To store the entire ancestry tree, the memory consumption is $O(AD \lg P)$. If $A \ll M$, $O(M)$ will be the dominating term. In RBGAF-SLAM, to store one single trajectory set, the memory consumption is $O(D \lg P)$; to store all the environment maps, the memory consumption is $O(nP)$. Now, $O(D \lg P)$ is negligible compared to $O(nP)$ since n is large.

In an environment where $A \ll M$, especially 3D space, $O(M)$ dominates $O(AD \lg P)$ in memory consumption. On the other hand, the actual number of objects observed within a 3D environment could be very few. For example, the resolution of current laser range finders measurements in distances is about 5 cm; therefore, a $400 \text{ m} \times 400 \text{ m} \times 20 \text{ m}$ grid map requires 2.56×10^{10} cells for one single map, which is impractical. The memory consumption of RBGAF-SLAM, instead, depends on the number of observations but not the dimension of the environment. Therefore, RBGAF-SLAM will be much more efficient in memory consumption in a 3D environment.

TABLE 4-2: COMPARISON BETWEEN SLAM ALGORITHMS

Algorithm	Raw Sensor Data	Model Sensor Noise	Posterior Estimation	Complexity	3D SLAM
EKF	NO	YES	Gaussian	$O(K^2)$	Good
FastSLAM	NO	YES	Gaussian	$O(\lg KM)$	Good
DP-SLAM	YES	YES	ANY	$O(APD \lg P)$	Poor
RBGAF-SLAM	YES	YES	ANY	$O(nPD \lg P)$	Good

Table 4-2 illustrates characteristics of major SLAM approaches. Feature based SLAM approaches are restricted by Gaussians while K is the number of features. FastSLAM is obvious more computational efficient compared to EKF. The computational complexity of feature based approaches is proportional to the number of features, not the scale of environment space. Therefore, it is suitable in 3D SLAM. However, it needs to control the number of features extracted from the 3D environment. RBGAF-SLAM is competing with DP-SLAM and obtain significant advantages in 3D view based SLAM. In RBGAF-SLAM, our analysis shows that the amortized complexity is $O(nP \lg P)$, compared to DP-SLAM $O(AP \lg P)$. For RBGAF-SLAM, $O(nP)$ memory space is required. Because of map resampling, A will be significantly larger than n at high resolution (relative to the sensor noise) and similar to n at low resolution, i.e. $A \geq n$. For example, if mapping an accurate range and bearing sensor data into low resolution map, one point measurement will be approximated by only one element, either a grid cell or a map chromosome. However, for high resolution or 3D SLAM, RBGAF-SLAM will present its advantages over DP-SLAM. In addition, 2D DP-SLAM is still near real time while RBGAF-SLAM achieves real time 3D SLAM with the aid of SIMD parallel computing, which will be introduced in the next chapter in detail.

4.5. Illustration of RBGAF-SLAM by a simple example

To illustrate RBGAF-SLAM process, a simple example is presented. Assume there is a robot with 2D laser range finders and there is a wall in front of the robot as presented in the figure 4-4 below.

Figure 4-4: Simple example of a robot moving towards a wall.

Assume there are only three Rao-Blackwellised particles ($P=3$) implemented although the actual implementation may have hundreds to thousands of Rao-Blackwellised particles. Assume the odometry is erroneous in bearing estimation but accurate in range estimation. Laser range finder has a range uncertainty of about 5cm.

4.5.1. Initialization

At the starting point, the robot position is set to (0,0,0), which stands for x,y and heading angle respectively. Therefore, it would be:

$$x_0^{(0)} = \{0,0,0\}, x_0^{(1)} = \{0,0,0\}, x_0^{(2)} = \{0,0,0\}$$

The first laser range finder scan is obtained. About 10 laser beams hitting on the wall in front of the robot. Therefore, the initial set of map chromosomes can be

generated. Let's generate one chromosome for each beam. There will be 10 map chromosomes associated to each Rao-Blackwellised particle. Therefore, there are altogether three Rao-Blackwellised particles, each particle contains two elements: a robot particle and a map $x_0^{(i)}, m_0^{(i)}$; each map contains 10 map chromosomes. Such three particles are presented as in figure 4-5.

Figure 4-5: Rao-Blackwellised particles at time $t=0$.

$$(a) \ x_0^{(0)}, m_0^{(0)} \quad (b) \ x_0^{(1)}, m_0^{(1)} \quad (c) \ x_0^{(2)}, m_0^{(2)}$$

These map chromosomes approximate the initial sensor scans and labelled by blue dots. The actual robot pose is marked in red while the robot pose in Rao-Blackwellised particle is marked in blue.

4.5.2. Motion update

After the robot move 1m forward, now it is next time interval so that $t=1$. The odometry shows the bearing angle is now 0.2 rad. With bearing angle θ , the new robot pose will be $x_1^{(i)} = \{1 \cdot \cos(\theta), 1 \cdot \sin(\theta), \theta\}$.

The actual bearing angle may be different from the odometry reading. Let's assume three possible bearing angles in 0.4, 0.2, and 0 rad, the actual bearing angle is assumed to be one of them. Up to now, the three Rao-Blackwellised particles will be estimated as shown in figure 4-6:

$$x_1^{(0)} = \{1 \cdot \cos(0.4), 1 \cdot \sin(0.4), 0.4\}, x_1^{(1)} = \{1 \cdot \cos(0.2), 1 \cdot \sin(0.2), 0.2\}, \text{ and } x_1^{(2)} = \{1 \cdot \cos(0), 1 \cdot \sin(0), 0\}.$$

Figure 4-6: Three robot particles estimated at time $t=1$.

$$(a) \ x_1^{(0)} \quad (b) \ x_1^{(1)} \quad (c) \ x_1^{(2)}$$

The actual robot pose should be one of these three robot particles as assumed. RBGAF-SLAM will update and find out the true robot pose from these three.

4.5.3. *Map update*

The second laser range finder scan is obtained. For the obstacles detected, a flag table is prepared. Now it is time to evaluate the map chromosomes. For the associated 10 map chromosomes, its relative range and bearing to the new Rao-Blackwellised particle $x_1^{(i)}$. The values are compared to the latest laser range finder readings. If range reading at that bearing angle is similar to the relative range, the flag of such bearing angle is marked and the weights for each map chromosomes are calculated. Randomly choose two map chromosomes and compare their weights. The winner will 'kill' the loser.

After the flag table is filled up and all map chromosomes are evaluated, new map chromosomes are generated and added into the map. Checking the flag table for all the bearing angles, if the flag is not marked but there is laser reading, generate a new map chromosome into the map with minor artificial noise (chromosome mutation).

The figure 4-7 demonstrates the map update procedures for Rao-Blackwellised particle $x_1^{(0)}, m_1^{(0)}$. In figure 4-8, it demonstrates the rest two Rao-Blackwellised particles $x_1^{(1)}, m_1^{(1)}$ and $x_1^{(2)}, m_1^{(2)}$ after map update process.

Figure 4-7: Map update for Rao-Blackwellised particle $x_1^{(0)}, m_1^{(0)}$ at time $t=1$.

Figure 4-8: Map update results for Rao-Blackwellised particles.

(a) $x_1^{(1)}, m_1^{(1)}$ and (b) $x_1^{(2)}, m_1^{(2)}$ at time $t=1$.

There are two maps based on ‘wrong’ pose estimation. Therefore, the flag table is not consistent to latest sensor readings. They will generate quite a lot new map chromosomes. Although the sensor observes the same obstacle, they ‘think’ the sensor observes new obstacle. On the other hand, for the map based on ‘correct’ pose estimation, the flag table is consistent with sensor readings and no new map chromosomes are generated. It keeps as 10 chromosomes without new chromosomes generated.

4.5.4. Trajectory update

The weight of each Rao-Blackwellised particle is calculated before the selection. For each Rao-Blackwellised particle, the weights time all time intervals are calculated individually. Take one robot pose $x_1^{(0)}$ as an example, with the associated chromosome map as presented in figure 4-9, the relative range and bearing measurement can be calculated based on the measurement model. The result is stored in a 1D array and such result is compared with the actual sensor

reading at $t=1$. If range from actual reading is longer, the weight $w_1^{(0)}$ scores a negative value, indicating that this is a “wrong” map chromosome. If two ranges are similar, it means the map chromosome is ‘correct’ and scores a positive value.

Altogether, it must calculate 6 weights:

$$w_0^{(0)}, w_0^{(1)}, w_0^{(2)}, w_1^{(0)}, w_1^{(1)}, w_1^{(2)}$$

The weights for the Rao-Blackwellised particles are:

$$w^{(0)} = w_0^{(0)} + w_1^{(0)}$$

$$w^{(1)} = w_0^{(1)} + w_1^{(1)}$$

$$w^{(2)} = w_0^{(2)} + w_1^{(2)}$$

As shown in figure 4-9, the particle $x^{(2)}$ has the highest weight, because the map and robot pose estimation satisfy with the sensor readings. Let’s randomly choose two, let $x^{(0)}$ and $x^{(2)}$ are chosen, since $w^{(2)} > w^{(0)}$, the tournament selection will cause:

$$x_0^{(0)} = x_0^{(2)}, x_1^{(0)} = x_1^{(2)} + N \text{ and } m^{(0)} = m^{(2)}$$

The winner particle will be copied to replace the loser particle with small mutation noise N , together with the associated map chromosomes.

Figure 4-9: tournament selection between Rao-Blackwellised particles $x^{(0)}$ and $x^{(2)}$.

If another two $x^{(0)}$ and $x^{(1)}$ are chosen, the result after selection is shown in figure 4-10. The ‘correct’ particle will kill the rest and replace the rest. As a result, the final particles will look the same and all represent the ‘correct’ robot trajectory with the ‘correct’ map approximated by map chromosomes. It shows that the actual robot pose at time $t=1$ should be $x_1^{(2)}$ and the environmental map should be approximated as $m_1^{(2)}$.

Figure 4-10: Three Rao-Blackwellised particles after RBGAF update at time $t=1$.

From the map update to the trajectory update, it completes a full cycle of RBGAF-SLAM update cycle. It keeps running for the following time intervals, the true robot trajectory and environmental map can be approximated.

4.6. Simulation and experimental results

4.6.1. Simulation: RBGAF-SLAM in 2D environment

The robot is assumed to be a two-wheel robot with the differential model. The odometry data is collected as the encoder reading and the gyro reading for every time interval. The measurement noise in both encoder reading and the gyro reading is assumed to be Gaussian.

For comparison with the DP-SLAM, the simulation environment is set to be the same as in [13]. The environmental map is shown in Figure 4-11. The map size is set as 24 by 16 meters.

Figure 4-11: Simulation environment for SLAM

The simulated robot is controlled by the component in Orca and travel in the simulation environment. The laser range finder data is collected and transferred to the RBGAF-SLAM module. The resultant environmental map is represented by a set of the chromosome maps. One of the chromosome maps obtained is shown in Figure 4-12. It can be observed that RBGAF-SLAM can recover the robot trajectory and map the environment.

Figure 4-12: RBGAF-SLAM result

Black: obstacles (chromosomes). Red: robot trajectory

Figure 4-13 and Figure 4-14 show the localization error of the RBGAF-SLAM results to the ground truth from the simulator.

Figure 4-13: Vertical axis: localization error in x (m) vs. horizontal axis: time (s)

Blue: position error. Pink: 3σ bound.

Figure 4-14: Vertical axis: localization error in y (m) vs. horizontal axis: time (s)

Blue: position error. Pink: 3σ bound.

The result is more accurate than the result in [13], with a much smaller number of particles. Compared to the DP-SLAM, both approaches use non-parametric methods for map representation. They can perform well in an unstructured environment without feature extraction. The major restriction is computational resources. With sufficient computational resources, both approaches can achieve good results. However, the computational resource is always limited. The result shown in [13] is processed by 24 hours of computing, which is not a practical approach. As presented in table 5-2, RBGAF-SLAM can achieve real-time speed, which is difficult for DP-SLAM.

4.6.2. Experiment: RBGAF-SLAM with 2D laser range finder

The first testing platform is the Segway robot. The robot has two tracked wheels with the encoders and gyroscope installed. The laser range finder is the SICK LMS as shown in Figure 4-15 and the experimental environment is in NTU block S2, as shown in Figure 4-16.

Figure 4-15: Segway two-wheel robot

Figure 4-16: Experimental environment at NTU-S2

This 2D laser range finder was recorded at a frequency of 30 Hz. The data is collected and processed by the RBGAF-SLAM. One of the resultant maps is shown in Figure 4-17. The obstacles are shown in black and the robot trajectory is shown in red.

Figure 4-17: Mapped environment

Red: robot trajectory. Black: obstacles.

The other testing platform is the packbot as shown in Figure 4-18 and the experimental environment is the NTU car-park F. The packbot is mounted with the LD-1000 laser range finder. The field of view is set to 270 degrees. With the encoder and the IMU readings, the RBGAF-SLAM result is shown in Figure 4-19 and Figure 4-20.

Figure 4-18: Packbot robot

Figure 4-19: RBGAF-SLAM result

Red: multiple robot trajectories. Blue: map chromosomes of single map.

Figure 4-20: SLAM result compared with real environment

The experimental results show that RBGAF-SLAM is working well with 2D laser range finders. The resultant environmental map matches the real environment.

4.6.3. *Experiments: RBGAF-SLAM with sonar sensor*

Here we apply RBGAF-SLAM in an underwater environment. In our experiment, the testing is conducted with the Blaze array sonar sensor, operating in the Singapore Reservoir. The sonar sensor also scans in a 2D plane, providing range and bearing information. However, unlike the 2D laser range finders, the sonar sensor also provides the signal strength against the time for the return signals. Therefore, the final scan data obtained can be represented in a 2D matrix instead of a 1D array. If the signal strength is plotted as the intensity value, the scanning results can be represented in a 2D image called a sonar image. Figure 4-21 below shows one example of the sonar image we obtained.

Figure 4-21: Underwater sonar image

The horizontal axis of the sonar image represents the bearing angle of the measurement. The vertical axis represents the range from the sensor position and the intensity value of the image represents the signal strength of the returns. The sonar sensor is less accurate than the laser range finder. It is difficult to implement reliable feature extraction algorithms because we don't know characteristic of obstacles in underwater environment. In our approaches, such characteristic is not needed. The noise model of the sonar sensor is assumed to be Gaussian with

standard deviation of 0.4m, obtained from experiments in swimming pool. It can be considered as a property related to the sensors, but not to the environment.

The sonar image is collected from the sonar sensor mounted beneath a boat. The boat is moving across the Singapore reservoir and detects some of the poles under the water. Unlike the 2D laser range finder, the data obtained from the sonar is a 2D matrix instead of the 1D array.

The motion of the boat is estimated by the DVL mounted under the boat. The GPS position is taken as the ground truth for comparison. The resultant map is shown in Figure 4-22 and the localization errors are shown in the Figure 4-23. It can be observed that, even with noisy data, RBGAF-SLAM still functions well without any feature extraction process over sonar images. Therefore, the difficulty of extracting reliable features from sonar images can be avoided.

Figure 4-22: RBGAF-SLAM result with sonar sensors

Red: robot trajectory obtained from SLAM. Green: ground truth from GPS. Black: mapped underwater objects.

Figure 4-23: Localization errors with sonar sensors

It can be observed that RBGAF-SLAM is capable of working with raw sonar images without feature extraction process. Although it is difficult to obtain the underwater ground truth, the GPS readings from the boat is taken as the ground truth for localization and compared with the SLAM results. It can be observed that the SLAM is consistent; localization errors are bounded.

4.6.4. *Velodyne 3D LiDAR experimental setups*

The RBGAF-SLAM approaches have the advantages of representing the environmental map more efficiently than the fine occupancy grid map, especially for a very large navigation space. If the environment is a 3D space, the number of grid cells required will be cubic in the size of the space. On the other hand, in the RBGAF-SLAM, the number of map particles needed to represent the same space is only quadratic in the size of the space. This chapter shows how 3D RBGAF-SLAM can be efficiently implemented even in 3D space in real time.

For 3D SLAM, 3D sensors are required to provide 3D environmental information. Monocular vision contains information from the 3D environment but does not provide range information. Stereo vision provides range information but the accuracy is relatively poor, especially in long-range measurements. Laser range finders provide accurate range measurements. Over the last decade, laser range finders have been proven to be accurate in range and bearing measurement compared to other sensors like vision or sonar sensors. Although many researches

have focused on the 2D laser range finder, the major drawback of the 2D LiDAR remains. The 2D LiDAR only provides 2D information for the environment while the actual environment is 3D. However, a 2D LiDAR sensor can be mechanically rotated to provide 3D range and bearing information with fairly good accuracy. The major drawback of such an approach is the updating time. Rotating the 2D laser range finder restricts the frequency of scanning the whole environment. However, robot navigation requires a fast response from sensors providing high data update frequencies.

The commercially available Velodyne 3D laser range finder mitigates this problem using multiple beams organized into layers while operating at 10~15 Hz, providing range and bearing measurements of more than one million points/second. The dense measurement data enables the detection of both static and moving obstacles in the 3D space; these advantages can significantly benefit mobile robot navigation. Therefore, the Velodyne has in the recent past attracted many robotics researchers, as in DARPA urban teams [61] and [62]. It has been proven to be an effective outdoor 3D perception device for mobile robot navigation in DARPA Grand Challenges and the DARPA Urban Challenge. The dense laser measurements provide for ease of detecting obstacles and tracking targets in a large 3D environment at reasonable operating speeds [63].

However, the dense measurement also requires large computational resources [64]. Due to such limitations, the current Velodyne processing algorithm must balance performance with computational cost to meet the requirements of real-time processing. These algorithms are restricted by the CPU speed. Researchers have developed simple and efficient processing algorithms for the Velodyne so that the data can be processed in real time in one CPU. More sophisticated processing algorithms, even though they would provide better performance, are not recommended since they require much greater computational resources. On the other hand, due to space and weight constraints, processing Velodyne data with multi-CPU's on the mobile platform is not a good choice. The challenge is that if the sensor provides dense 3D measurement, it will generate better results but requires more computational resources. SLAM, especially, is computationally expensive and

requires a lot of computational resources to achieve real-time speed. It is even more difficult for 3D SLAM with a high data rate.

To perform SLAM in unstructured urban and off-road 3D environments is complex and challenging. Dust, heavy rain, platform vibrations, and object reflectivity all affect the performance characteristics of the 3D sensors, and the slope of the ground and vegetation/jungle makes the navigation task still more complicated. Further, the dense 3D data could be complex and difficult for feature extraction. On the other hand, occupancy grid based approaches, like DP-SLAM, will be computationally expensive in a 3D environment since their computational cost is cubic in the environment size.

RBGAF-SLAM, instead, works with a compressed chromosome map. The accuracy of the 3D LiDAR is relatively good (error less than 0.3 m). Therefore, a compressed chromosome map actually represents ‘surfaces’ in a 3D environment. The computational resources required to represent the 3D environment in a compressed chromosome map are significantly less than for an occupancy grid map. RBGAF-SLAM should demonstrate more advantages in a 3D environment compared to the occupancy grid based approaches.

To enable RBGAF-SLAM with high resolution 3D LiDAR (the Velodyne) and achieve real-time processing on a space/weight limited platform, we developed ultra speed RBGAF-SLAM based on GPU processing and the Compute Unified Device Architecture (CUDA). The Velodyne data processing and the RBGAF-SLAM are done in a highly parallel architecture so that it can run on commercially available GPUs. The processing time is significantly reduced. As a result, 3D RBGAF-SLAM can be achieved in real time on a space/weight limited system.

Besides, with 3D RBGAF-SLAM, the data collected from 3D LiDAR can be processed and output as 2D results, e.g., a 2D obstacle map. In this way, the 3D Velodyne LiDAR can be used as a 2D sensor and yield 2D results. Compared to normal 2D sensors, such 2D results are much more reliable since they are extracted from informative 3D data. For example, 2D laser range finders only scan a plane. Obstacles higher or lower than this plane will not be detected. The obstacle map

obtained from 3D data, on the other hand, will not have this problem. Although such compression will lose some 3D information, 2D SLAM, anyway, is much faster than 3D SLAM. Therefore, it is one useful option for RBGAF-SLAM and robot navigation. Such processing algorithms are introduced in this chapter as well.

4.6.5. *Experimental result with Velodyne 3D LiDAR*

The experiment is conducted with the Velodyne 3D LiDAR mounted on the top of a SUV vehicle. The uncertainty of the Velodyne range measurement is estimated through calibration to be 0.2 meters. The vehicle is equipped with GPS and its reading is taken as the ground truth for verification.

The experiments for 3D SLAM are conducted in several places with the same Velodyne sensor. The vertical field of view is about 26 degrees, from approximately +2 to -24 degrees. The Velodyne 3D LiDAR is mounted on top of the vehicle so that it can detect the ground and nearby objects like lamp-posts, trees, buildings, etc. The RBGAF-SLAM does not have any prior knowledge about the environment, except the measurement uncertainty of the Velodyne. The single Velodyne beam is fairly accurate. However, the multi-beam measurements are not so accurate – the range measurement uncertainty is about 0.2 to 0.3 meters. This has been obtained from our calibration test and also mentioned in the literature. The uncertainty of the bearing/pitch measurement is assumed to be 1 degree. Unlike feature-based approaches, prior knowledge from the environment is not needed. In fact, in a different environment, our tests use the same model.

Two experiments are completed on the NTU campus and one in an off-road (jungle) terrain. To fully verify the performance, efficiency and effectiveness of 3D RBGAF-SLAM in different situations, varieties of tests with artificial sensor noises are implemented. These virtual tests fall into four categories based on the first NTU trial.

- With different proposal distributions

The proposal distribution is usually obtained from odometry readings. Here, results obtained from experimental data with large virtual noises are presented to verify the effectiveness of RBGAF-SLAM. In addition, it also evaluates the improvements by matching latest two scans. If RBGAF-SLAM works well with large virtual noises, it would perform better with less noise as in real implementations.

- With different chromosome sizes

This is an important parameter to be determined. The value is directly related to the computational cost. It demonstrates the number of map chromosomes generated and maintained in RBGAF-SLAM with different chromosome sizes. It demonstrates the effectiveness of GAF resampling, especially at high resolution.

- With different virtual measurement noise

This is to verify the effectiveness for different sensors. Our experiments are conducted with laser range finders which are more accurate than sonar or stereo vision. With different virtual measurement noise, it presents that RBGAF-SLAM should still be effective with other bearing and range sensors of less accuracy.

- With different Velodyne processing algorithms

Velodyne 3D measurements can be further processed for SLAM. This presents a possible way to balance SLAM performances and processing speed by using processed map.

4.6.5.1. *Experimental results with different proposal distributions*

The usual proposal distribution in SLAM is created by the robot motion model. However, the uncertainty is relatively large and requires a large number of particles (trajectory/map pair) to recover the true robot trajectory. This will result in a large computational cost and, quite often, it is difficult to achieve real-time speed. In FastSLAM2 [29], it is pointed out that the laser scans can also be used to generate more accurate proposal distributions. The approach is to use the latest two laser scans and scan matching approaches to estimate the robot motion. The

computational cost of this process is relatively low compared to resampling process of the Rao-Blackwellized particle filter, but it can significantly improve the performance. This idea is also applied in DP-SLAM: 2.0 [65] and demonstrates good performances.

The reason that such an approach can improve the performance is that the accuracy of the scan matching is far better than the motion model. Therefore, when the odometry/motion model is poor in accuracy, it is wise to apply other approaches to give better proposal distribution. This will significantly reduce the computational cost, or, on the other hand, improve the performance for the same computational cost. Here, tests are designed to compare the 3D GAF-SLAM performances under different proposal distributions.

The trajectory recorded by GPS is taken as the ground truth. Due to the hardware limitations, it is difficult to conduct experiments on all types of robot platforms. Therefore, artificial sensor noise is added into the recorded data to simulate different robots with different odometry uncertainty. Two proposal distributions are tested. The first is the one with artificial noise and the second one is filtered by the latest two laser scans.

The first test gives a qualitative verification of RBGAF-SLAM in heading. The heading noise is assumed to be very large and only 50 particles (trajectories/map pair) are used. It has a large bias error at 15 degrees per second and the motion model estimates the uncertainty of the heading rate to be 25 degrees per second. In Figure 4-24, the ground truth is shown. The obstacles (map chromosomes) are shown in black/grey color. The darkness of the dot indicates the height. The robot trajectory is represented in red. RBGAF-SLAM contains multiple chromosome maps and multiple robot trajectories, but all the robot trajectories with only one map are shown. The uncertainty of the robot trajectories is shown in Figure 4-25. It can be observed that the mapping result is completely unacceptable due to the large heading errors.

Figure 4-24: Ground truth and mapping result (45 degree view)

Red: robot trajectories. Black: map chromosomes.

Figure 4-25: Mapping result with large heading noise (45 degree view)

Red: robot trajectories. Black: map chromosomes.

If the robot poses are filtered by the latest two laser scans, it can be improved. At every interval, the best 30% of the poses are reserved and the rest are rejected. The result is shown in Figure 4-26.

Figure 4-26: Trajectories filtered by latest two laser scans (45 degree view)

Red: robot trajectories. Black: map chromosomes.

With RBGAF-SLAM, the result is much better. Figure 4-27 demonstrates the result. In the Figure 4-28, the mean heading error and the 3σ bound are shown.

Figure 4-27: RBGAF-SLAM result with very large heading noise (45 degree view)

Red: robot trajectories. Black: map chromosomes.

Figure 4-28: Heading errors of RBGAF-SLAM test one

The result shows that with very large heading noise and only 50 particles, the RBGAF-SLAM is still functioning and provides meaningful results. However, since the heading noise is too large, to achieve a good SLAM result, 50 particles are not enough.

The inaccuracy of a realistic gyroscope cannot be as large as 15 degrees per second. To quantify the 3D RBGAF-SLAM performance, the second test is implemented. The heading noise is assumed to be 2 degrees/min, which is a reasonable drift of a cheap gyroscope. The noise in encoder reading is set to be 5% of the distance travelled. With these noises, the RBGAF-SLAM results are shown below in Figure 4-29 and Figure 4-30. The vehicle starts and stops almost at the same location. Figure 4-31 presents the trajectory from the odometry used in the test.

Figure 4-29: RBGAF-SLAM result (in NTU campus, top view)

Red: robot trajectories. Black: map chromosomes.

Figure 4-30: RBGAF-SLAM test result (45 degree view)

Red: robot trajectories. Black: map chromosomes.

Figure 4-31: Trajectory from the odometry readings used in SLAM

Red: robot trajectories. Black: map chromosomes.

The localization errors are shown in Figures 7-32 and 7-33 with the 3σ bound.

Figure 4-32: Localization error in x axis

Figure 4-33: Localization error in y axis

It can be seen that RBGAF-SLAM performs well in this test. The time required for GAF map update is less than 50 ms, which is quite small. The updating time for the full Rao-Blackwellized particle is about 0.6 seconds or 600 ms on the NVIDIA

GT250 graphics card. The overall processing time is less than the test duration so that real-time speed can be achieved.

4.6.5.2. *Experimental results with different chromosome sizes*

Similar to the grid cell size, the virtual chromosome size determines the map resolution. The smaller the size, the higher the map resolution. Higher resolution results in better approximation accuracy and can provide more accurate mapping results. However, higher resolution requires much greater computational resources. With limited computational resources, high resolution will result in fewer particle numbers (trajectory/map pair). This has a negative effect on the SLAM performance. Therefore, the map resolution is not always ‘the higher, the better’.

TABLE 4-3: SLAM RESULTS WITH VARIOUS VIRTUAL CHROMOSOME SIZES

Virtual chromosome size (m)	0.1	0.2	0.4	0.6
Mean localization error (m)	0.19	0.21	0.46	0.55
Number of chromosomes	220k	90k	72k	64k

In Table 4-3, the mean localization errors with various virtual chromosome sizes are shown. The computational cost can be considered to be proportional to the number of map chromosomes. It can be observed that the virtual chromosome size can improve the localization accuracy. However, when the virtual chromosome size is less than 0.2 meters, this improvement is not significant, compared to the increase in computational cost.

4.6.5.3. *Experimental results with different measurement noise*

TABLE 4-4: LOCALIZATION ERRORS WITH VIRTUAL MEASUREMENT NOISES

Measurement noise (m)	0.4	0.8	1.2
Mean localization error	0.21	1.03	1.32

Here, the Velodyne 3D LiDAR reading is added with artificial noises. With such noisy 3D data, the RBGAF-SLAM performances are shown in Table 4-4. It can be

seen that better sensors always provide better results. However, if the sensor is more accurate, it requires smaller virtual chromosome size to optimize the performance. In such a case, a reduction in virtual chromosome size is necessary.

4.6.5.4. *Experimental results with different Velodyne processing algorithms*

Here, the Velodyne 3D LiDAR reading is processed and results in a 2D output. The obstacle/ground classification will generate the obstacle map and such obstacle map can provide 2D range and bearing measurements. Unlike the results obtained from a 2D laser range finder which can only scan obstacles in a flat plane, such 2D range and bearing measurements are derived from 3D data so that it is less accurate but more reliable. It can project obstacles higher or lower than a 2D horizontal scanning plane into a 2D map. For example, the SLAM result with 2D obstacle map is shown in the Figure 4-34. Besides, the curb map generated from the Velodyne data can be used for SLAM as well, though it is not as informative as the obstacle map.

Figure 4-34: RBGAF-SLAM result with 2D obstacle map (top view)

Red: robot trajectories. Black: map chromosomes.

With such processed data, the 2D RBGAF-SLAM performances are shown in Table 4-4. It can be seen that 2D SLAM with obstacle map is more accurate than that with curb map. Compare to the 3D laser point clouds, the 2D data derived is less informative, less accurate, noisier, but the computational cost is lower. However, 2D SLAM does not correct pitching errors so that it can only perform as a substitute when the computational resources are very limited.

Table 4-4: Localization errors with different Velodyne processing algorithms

SLAM data	3D point clouds	2D obstacle map	2D curb map
Mean localization error (m)	0.21	0.94	2.56
Number of chromosomes per map	>90000	2323	7638

4.7. Improvements for RBGAF-SLAM

RBGAF-SLAM can be further improved in two aspects: RBGAF-SLAM in a dynamic environment, and genetic learning with test data.

4.7.1. Dynamic RBGAF-SLAM

The standard RBGAF-SLAM is developed under the assumption that the environment is static. The moving (dynamic) objects can be filtered out by using the negative sensor information during the mapping. If there are objects moving in the environment, it will appear in the chromosome maps and vanish after some time, filtered out by the map resampling process. This phenomenon is similar to what happens in occupancy grid mapping. However, this filtering only occurs after the objects are mapped. It still causes noise in the SLAM and the map will contain noises caused by the moving objects, as presented in Figure 3-24. To extend RBGAF-SLAM into a dynamic environment and filter out the moving objects/spurious noise before they are mapped, we choose to implement pre-

processing on sensor inputs [66] by using the past (recorded) negative sensor information. In this way, the dynamic objects/spurious noise will be recognized and filtered out from the environmental maps before it is used in standard RBGAF-SLAM.

4.7.1.1. *Pre-processing and sensor rejection*

RBGAF-SLAM records the full robot posterior and each map has one corresponding robot trajectory. Therefore, the past negative sensor information can be collected based on the robot trajectory and sensor measurement records. If at all previous time instance $t = 0, \dots, t-1$ measurements and robot poses are such that there are no objects at a location, we would have confidence to conclude that at time t , if an object is observed at that location, it must be either sparse noise or a dynamic object.

The pre-processing is implemented in the map chromosome generation process as described in Section 3.3.4. The weight of each new map chromosome generated $m_t^{(i,j)}$ will be evaluated with the robot trajectory $x_{0:t-1}^{(i)}$ and recorded sensor measurement $z_{0:t-1}$. A positive weight means obstacles have been observed and should be compressed; a negative value indicates that according to previous measurements it is empty space; zero means no conclusions can be made since it is an unknown space.

In Section 3.3.5, the evaluation of a map chromosome $m_t^{(i,j)}$ with robot pose $x_t^{(i)}$ and sensor measurement z_t is

$$f_{map}^{(i,j)}(m_t^{(i,j)}, x_t^{(i)}, z_t) = {}^m w_t^{(i,j)} = V(z_t, l_t^{(i,j)}, h_t^{(i,j)}) = M(r_t^{(n)}, \alpha_t^{(n)}, l_t^{(i,j)}, h_t^{(i,j)})$$

Similarly, the pre-processing weight ${}^p w_t^{(i,j)}$ of each new map chromosome generated $m_t^{(i,j)}$ will be evaluated with the robot trajectory $x_{0:t-1}^{(i)}$ and recorded sensor measurement $z_{0:t-1}$.

$${}^p w_t^{(i,j)} = \sum_{k=0}^{t-1} V(z_k, l_k^{(i,j)}, h_k^{(i,j)}) = \sum_{k=0}^{t-1} M(r_k^{(n)}, \alpha_k^{(n)}, l_k^{(i,j)}, h_k^{(i,j)})$$

If the resultant weight is positive or zero, it indicates that the new map chromosome should be a static object. If the weight is negative, the new observation should be considered a dynamic object and get rejected from RBGAF-SLAM after the chromosome generation.

4.7.1.2. *Computational complexity for dynamic RBGAF-SLAM*

Similar to the discussions in Section 4.4, in a dynamic environment, let the area of one single occupancy grid cell (or the area covered by a square of size $\varepsilon \times \varepsilon$ in RBGAF-SLAM) be 1, the total area in the sensor measurement range be M , the area swept by the sensors (obstacles) be A , area with negative sensor information (free/empty space) be B and unknown space be C . Then $M = A + B + C$. Normally, it can be assumed that $B \gg A$ and $C \gg A$, i.e., both B and C are much larger than A .

If both static objects and dynamic objects are observed, let the observation caused by static objects be a_1 and the observation caused by dynamic objects/spurious noise be a_2 , then obviously $A = a_1 + a_2$.

In occupancy grid maps, at any time instant t , to recover the negative sensor information, B cells must be processed: $O(B)$. The total computational cost from $T=0$ to t is $O(tB)$. For our approach, to recover the negative sensor information with a depth t , the complexity will be $O(ta_2)$. Since $tB \gg tA > ta_2$, therefore, our approach will be much more efficient than an occupancy grid map in dynamic environment. With high resolution sensors and a high resolution occupancy map, the $\frac{B}{a_2}$ ratio could be extremely high. For example, with 5 cm resolution, a 50 m x 50 m environment contains 1,000,000 cells and most of them are unknown space or free space. At the same time, the dynamic objects to be identified can be very few

and the $\frac{B}{a_2}$ ratio could be as high as several tens of thousand.

4.7.1.3. *Experimental result for pre-processing*

As shown in Figure 4-35, the mapping result of the dynamic RBGAF-SLAM is compared with the result shown in Figure 3-23. It can be seen from Figure 4-35 that the moving car which can be observed in standard GAF mapping does not appear with pre-processing. The mapping result with pre-processing is clearer.

Figure 4-35: Experimental result of pre-processing and sensor rejection

(a) Standard GAF mapping result. (b) mapping result from dynamic RBGAF-SLAM. The moving car does not appear.

4.7.2. *Genetic learning for optimizing RBGAF-SLAM parameters*

As shown in this chapter, Section 4.3, the memory efficiency of RBGAF-SLAM is much better than that of DP-SLAM with occupancy grid maps. It does not rely on feature extraction so that it does not require prior knowledge about complex unstructured environment. However, if we have some prior knowledge, SLAM approaches can be tested in simulations/experiments with similar characteristics of the real environment to optimize the performances. Here, RBGAF-SLAM can also benefit from such tests. If test data sets can be collected, RBGAF-SLAM can be further optimized.

The well known optimization approaches, expectation maximization (EM) strategies [23], can be used to optimize the parameter set for the RBGAF-SLAM approach. However, EM theory suffers from the local maximum problem. The successful implementation of EM theory depends on a good initial ‘guess’. Due to the high complexity of the RBGAF-SLAM problem, the availability of a good initial ‘guess’ is not guaranteed. Implementing EM theory and searching for the proper parameter set may lead to a success provided one has access to good training/testing data. However, genetic algorithms have proven to be a good global optimization tool for a number of applications. It does not require a good initial data set to find out the global optimum. Therefore, the genetic learning approach is selected here for optimizing RBGAF-SLAM. It can optimize RBGAF-SLAM in several aspects: the motion modelling, the resampling frequency and the genetic algorithm parameters. These aspects are explained in what follows.

4.7.2.1. *Improper modelling*

During the implementation of RBGAF-SLAM, it was discovered that an improper motion estimation model may lead to inconsistent results. Although RBGAF-SLAM enables the use of any motion estimation model or sensor measurement model, they would never perform well with an erroneous model. When the uncertainty of the motion estimation is under-estimated, consistency might be lost and SLAM will fail. To avoid this, we need to assume a larger uncertainty of motion estimation in SLAM. However, such over-estimated uncertainty would reduce the SLAM accuracy and also requires a much larger number of Rao–Blackwellized particles to cover. The increase in the number of Rao–Blackwellized particles will increase the computational cost, both in memory consumption and computational time. Therefore, it is necessary to determine the proper parameters of motion estimation model through experiments.

4.7.2.2. *Resampling frequency*

For RBGAF-SLAM the choice of resampling frequency is very important. If the resampling frequency is too high, the sample impoverishment problem will affect the accuracy. RBGAF-SLAM can mitigate the problem and be less sensitive to the

resampling frequency. However, an optimized resampling frequency still benefits RBGAF-SLAM. As shown in Figure 4-36, tournament selection is generally better than Monte Carlo resampling and much less sensitive than resampling frequency.

Figure 4-36: Influence of selection method and resampling frequency

Elitism tournament selection (Tournament2) shows different behaviour compared to the standard tournament selection, but the difference is not significant.

4.7.2.3. Genetic algorithm parameters

There are other genetic algorithm parameters which can be optimized, including the mutation rate, mutation range, and crossover rate. Therefore, if a testing data set is available, it is necessary to optimize these parameters so that RBGAF-SLAM will be able to perform better in the real environment.

From Section 4.7.2.1 to 4.7.2.3, we addressed the above three major issues about RBGAF-SLAM. These problems can be solved by increasing the number of Rao–Blackwellized particles, because a large amount of Rao–Blackwellized particles can significantly improve the accuracy and robustness of the SLAM approach. However, the number of Rao–Blackwellized particles is strictly limited due to the computational resources. In our approach, offline genetic learning techniques are implemented to search for a proper set of RBGAF-SLAM parameters.

4.7.2.4. Genetic learning algorithms

Figure 4-37: Genetic learning algorithm for RBGAF-SLAM

To determine the GA techniques and corresponding parameters, we designed a genetic learning approach for RBGAF-SLAM. Its overall design is shown in Figure 4-37.

4.7.2.5. Genetic encoding

Each genetic code is designed as follows:

$$L_Code_n = \{M1 | P1, M2 | P2, M3 | P3, \dots\}, n = 1, 2, 3, \dots, N$$

L_Code_n is the code for genetic learning. The ‘M’ stands for the operators/methods chosen in RBGAF-SLAM, e.g. tournament selection, mutation, etc. Each operator is one integer value representing one GA technique or a method name. P is a floating point value (real value) to represent GA parameters related to the corresponding operator. Since more than one method is used, the genetic codes

consist of a list of methods M with corresponding parameters P . The program then reads the genetic code and implements the code by calling the corresponding function block. The parameters to be determined for the motion estimation model are also encoded into these chromosomes.

The fitness function is designed to evaluate the performance of the SLAM results. This criterion is very important for the success of the genetic learning approach and depends on the implementation objectives. In our approach, SLAM consistency and the localization errors are used to evaluate the ‘goodness’ of the SLAM results.

For evaluating SLAM approaches, consistency is the first priority. When the robot is passing through a position, the diversity of all the robot particles (trajectories) must include the true robot position. If the true robot position is outside the spread of the particle set, it should be considered a failure for the SLAM approach. The weight of the error is calculated by the minimum Euclidean distance from a map chromosome m to a particle at time t (robot pose). To penalize the inconsistency, a constant error c is added into the result. Here, c is set at 2 (meters).

$$Err_t^{(i)} = c + \sum_{j=0}^N \min(dist(m_j^{(i)}, x_t^{(i)}))$$

If consistency is guaranteed, the localization error is chosen as the second criterion to evaluate the SLAM results. The ground truth is collected at several chosen points and the localization errors are calculated for the fitness function.

$$Err_t^{(i)} = \sum_{j=0}^N \min(dist(m_j^{(i)}, x_t^{(i)}))$$

The consistency and the accuracy of SLAM results are all taken into account in the fitness function. The final fitness value is calculated by summing the weights at different time instances.

$$Err^{(i)} = \sum_{T=0}^t Err_T^{(i)}$$

The learning process uses tournament selection to find the best-fit combination of GA operator for RBGAF-SLAM. In our implementations, SLAM accuracy is evaluated by the mean square localization error or loop closing error (if it is difficult to obtain the ground truth of the robot trajectory).

Genetic learning itself also contains some undetermined parameters. However, unlike SLAM, genetic learning can be implemented and processed offline. More importantly, there is generally no time constraint on the processing. Even if the genetic learning parameters chosen are not optimal, the optimized SLAM results are still acceptable. The problem of choosing genetic learning parameters is not critical.

In standard GA, one point crossover operation takes place at a random point in the genetic codes. However, due to the tight combination of methods and parameters in our genetic codes, it is not meaningful to cut them apart. Our genetic learning is designed in such a way that all methods and corresponding parameters are combined as one element in the crossover process. The process will not separate parameters and methods, so the genetic codes are always meaningful to the RBGAF-SLAM.

TABLE 4-5: GENETIC CODE FOR OPTIMIZING RBGAF-SLAM

	Methods	Options	Parameters
M1	Selection methods	Tournament, scale, ranking	N.A.
M2	Mutation	N.A.	mutation range, mutation rate
M3	Crossover	N.A.	crossover rate
M4	Resampling frequency	N.A.	frequency
M5	Motion estimation model	N.A.	distance uncertainty, heading uncertainty

Besides, mutation operations take place on both methods M and parameters P. The

mutation range of the parameters P is proportional to the original value of that parameter. The mutation operator on the method is to choose a random method in the same category. For example, in the category of tournament selection there can be ranking selection and scale selection, but not a mutation method. Otherwise, it will be impossible to implement the genetic code for RBGAF-SLAM. Table 4-5 shows the details of the genetic learning code.

The genetic learning approach is summarized as follows:

Genetic_learning(pm, rm, pc):

For all genetic codes L_Code_n

{

For all $P_i, i = 1, 2, 3, 4, 5$, draw random value $pr \in (0, 1)$, if $pr < pm$,
 $P_i = P_i * (1 + rm)$

With probability of pc {Draw two samples $L_Code_a.M_b$ and $L_Code_c.M_d$:
 replace by $L_Code_a.M_d$ and $L_Code_c.M_b$ }

fitness_function($RBGAF - SLAM(L_Code_n)$)

}

Tournament_selection($L_Code_n, n = 1, 2, \dots, N$)

Repeat for next generation.

4.7.2.6. Genetic learning results

Figure 4-38 shows the improvement from the initial GA codes to the final GA codes obtained through the genetic learning process. The test data and simulation setup is the same as presented in Section 4.2.1. The RBGAF-SLAM performances of GA codes are evaluated by the mean loop closing error.

Figure 4-38: Mean loop-closing error through genetic learning

It is obvious from the figure that GA learning reduces the mean loop closing error generation after generation. The best GA code provides a loop closing error of 0.17 meter which is better than the initial performance.

4.8. Summary

This chapter presented the proposed RBGAF-SLAM. This is an efficient means of localizing the mobile robot in an unknown environment, and at the same time, mapping the environment.

The basic concept of the proposed RBGAF-SLAM approach is to combine particle map, Rao–Blackwellized particle filter and the genetic algorithmic filter (GAF) for SLAM. The GAF mapping approach presented in chapter 3 is major procedure and pre-requisite to the success of RBGAF-SLAM. Rao–Blackwellized particle filter provides sound mathematical foundation for this approach to solve SLAM problem. The approach is then effective in a highly complex environment with good computational efficiency. The Rao–Blackwellized particle filter provides the theoretic foundation of the SLAM approach and genetic algorithmic techniques are implemented to mitigating the shortcomings of a particle map.

Although an arbitrary motion estimation model or inverse sensor measurement

model can be applied to the RBGAF-SLAM framework, this chapter presented a detailed formulation for one example of RBGAF-SLAM implementation in 2D space. A variant of RBGAF-SLAM was also presented, including the back-propagated estimation of the map chromosomes for the dynamic environment, and a genetic learning approach for optimizing the RBGAF-SLAM parameters. The computational complexity was also briefly discussed and compared with other SLAM approaches.

It was shown that RBGAF-SLAM is an effective and efficient view-based SLAM approach in different environments, with different sensors. The 3D SLAM results verify that RBGAF-SLAM is an efficient and effective Rao-Blackwellized particle filter based SLAM algorithm. It does not require prior knowledge for feature extraction compared to the feature based SLAM approaches.

For RBGAF-SLAM, performance can be improved by matching latest two 3D scans. The experimental results shows that RBGAF-SLAM can still provide satisfactory results with poor odometry readings or less accurate 3D bearing and range sensors. The computational cost of DP-SLAM will increase significantly with high map resolution. For RBGAF-SLAM, such increase is much smoother. Compared to DP-SLAM, RBGAF-SLAM is more efficient, especially in high resolution and 3D SLAM.

CHAPTER V: HIGH SPEED RBGAF-SLAM

5.1. Motivation

Mobile robot SLAM requires real-time processing. This indicates at least two basic requirements for the SLAM algorithms. First, at any time t , the algorithm only requires input data collected at or before time t . If the algorithm requires data collected after time t to get the results, it can only be applied off-line.

Secondly, the frequency of the SLAM updates is high enough. The minimum frequencies for different applications are quite different. When the robot is driving fast and the sensor readings are noisy, it requires a higher updating frequency to guarantee localization accuracy. When the robot is driving at very low speed, the required updating frequency could be very low. It is hard to set a threshold for the term ‘real time’. Therefore, in our experience, in this research, we believe that if the updating time is less than 300ms (>3 Hz), it can be marginally considered as ‘real time’.

However, the actual processing frequency depends on the computational resources. Therefore, an algorithm that achieves real time in the lab may not achieve real time on the robot. Due to limitations like space, weight, power consumption and cost, the computational resources available on a robot platform are always limited. Since the computational resources on different robot platforms vary a lot, we take a desktop PC as the benchmark device. There are several reasons for this.

First, desktop PCs are relatively cheap and affordable for most researchers /companies.

Secondly, the desktop PC has a number of equivalent substitutes with similar computational resources and other advantages. For example, a laptop PC could be as fast as a desktop PC but much smaller and lighter. There are some commercial

solutions like CompactPCI, providing similar computational resources but much more reliable and smaller in size. These substitutes can satisfy different robotics platforms.

Thirdly, the software developed for desktop PCs can be transformed into other platforms.

When the computational resources of SLAM approaches are considered, the feature-based approaches and view approaches perform differently.

For feature approaches, with given feature extraction/filtering algorithms, the number of features extracted is fixed and the performance can be expected to provide the number of features, so the computational cost of standard EKF SLAM can be estimated. FastSLAM, on the other hand, can adapt to the computational resources. It can improve its performance by increasing the particle number, but it will not change the number of features. If reliable features can be extracted, the performance and efficiency will be good. If very few reliable features are extracted, increasing the particle number does not help the map representation.

For view based approaches, like grid map based DP-SLAM and RBGAF-SLAM, the resolution of the map representation plays an important role. With higher map resolution, the map representation will be more accurate and better SLAM performances can be expected. More importantly, a high resolution enables the implementation of more sophisticated sensor models. It also improves the robustness of representing a complex unstructured environment. Similar to FastSLAM, both DP-SLAM and RBGAF-SLAM can vary the number of particles (robot trajectory). Therefore, they are more flexible in adapting to the computational resources. If more computational resources are available, they can always benefit. However, higher resolution requires more computational resources. Running the DP-SLAM on a normal desktop PC is 'near real time' [13]. To achieve excellent results takes many hours of processing. Although the codes can be optimized, both DP-SLAM and RBGAF-SLAM require more computational resources to achieve real time with high resolution map representations.

Due to the rapid advance of IC technology, CPU speed has been doubling every 18 months. Therefore, a processing speed that has improved by 100% is a good achievement in applications, but not significant as a research contribution. However, speed accelerated by more than 10 times indicates that the robot will be able to use processors developed 4~6 years in the future. Such progress is important and it is worth investigating such accelerating approaches. It is the intention in this chapter to demonstrate how the processing speed of the proposed algorithm can be significantly improved for real-time performance using parallel computing architectures available in modern GPUs.

The rest of the chapter is organized as follows: Section 5.2 gives an overview of the related background about parallel computing and Compute Unified Device Architecture (CUDA); Section 5.3 describes the highly parallel architecture for proposed RBGAF-SLAM in detail; Section 5.4 gives experimental results to demonstrate the significant improvement in processing speed. It guarantees the real time processing of RBGAF-SLAM. The end of this chapter is a summary.

5.2. Background knowledge about CUDA

5.2.1. GPU and Compute Unified Device Architecture (CUDA)

CUDA is a parallel computing architecture developed by NVIDIA [67]. More specifically, CUDA is the computing engine in NVIDIA graphics processing units or GPUs that is available to software developers through standard programming languages such as C.

This general purpose parallel computing architecture leverages on the parallel compute engine in NVIDIA graphics processing units (GPUs) to execute many complex computational problems in a fraction of the time that it would take to execute on a CPU based configuration. To program in the CUDA architecture, developers today can use C, one of the most widely used high-level programming languages, which can then be run at great performance on a CUDA enabled

processor. With the help of CUDA, GPU can be used in mobile robot navigation and achieve a huge leap in system performance. For example, in [68], CUDA is implemented for tracking. With the aid of CUDA GPU processing, real time target tracking is achieved.

5.2.2. GPU parallel computing: Advantages

The availability of GPU programming tools enables us to use the computational power of the GPU, which is far faster than a normal CPU in a PC. Compared to specially designed processors, GPUs are much cheaper and easily available. Although GPUs are not designed for mobile robot navigation, they have been developed over decades and are produced in large quantities today. The demand for high-end gaming has been pushing GPU performance higher and higher since 2000. Figure 5-1 shows the computational power of NVIDIA GPUs against the Intel processor from 2001 to 2008.

Figure 5-1: Processing speed of GPU vs. CPU

Figure 5-1 clearly shows that the GPU is becoming a more and more powerful computing device. Table 5-1 below shows the price (recorded in 2009 August) of GPU cards against CPUs, with the performance/price ratio presented in Figure 5-2. It can be seen that the GPU has obvious advantages in the performance/price ratio.

TABLE 5-1: COMPUTATIONAL POWER FOR CPU & GPU

Processor	Speed(GFLOPS)	Price(\$\$)	P/P ratio (MFLOPS/\$\$)
C2D 8400	~24	257	93.38521
G9800GT	~500	209	2392.344
GTX280	~933	605	1471.609

Figure 5-2: Performance / price ratio for CPU & GPU

The performance of the top GPU today is about 20~40 times faster than the CPU. The performance/price ratio is even better for most GPU cards. The performance/weight, performance/volume and performance/power consumption ratios are all much better as well. Moore's well known theory says that processor speed will double every 18 months. If a GPU can be used to implement navigation algorithms, the performance will be dramatically better than that of a CPU, or say, equivalent to the performance achieved by a CPU developed six to ten years in the future.

Besides the NVIDIA GForce 8, 9, 200 and Tesla series, Intel also announced their Larrabee GPU [69] which is scheduled in 2012. The Larrabee processors are supposed to have 16 cores in one chip and can process 64 threads in parallel.

Although the Larrabee GPU is not available on the market today, we can foresee a bright future for high performance computing capability.

A GPU card is about the same size and weight as a compact PC card. If the GPU is used to process the Velodyne data, the system can achieve 20~40 times speed boost up with similar space and weight. For mobile robot navigation, this significant improvement would make those non-real-time algorithms into real-time algorithms. In the future, if 3D laser range finders provide a higher data rate, the GPU can still process them in real time. It is worth re-writing/modifying/optimizing the Velodyne processing algorithms to satisfy the GPU requirements.

5.2.3. *GPU parallel computing: Limitations*

However, to make full use of the computational power of the GPU is not easy. Currently, most PC software does not support general purpose GPU computing. A number of requirements have to be solved.

- Scalable highly parallel computing

GPU computing requires a parallel computing algorithm. Unlike multi-core CPUs, GPUs generally run a large number of threads, usually in terms of hundreds. A normal multi-core CPU, however, only requires two to eight threads to make use of the computational resources. The number of threads must be highly scalable for GPU implementations; otherwise, the computational resource might be wasted. To achieve this, the parallel algorithms must be able to be split into hundreds to thousands of threads.

- SIMD processors

GPU processors run in SIMD mode, i.e. Single Instruction Multiple Data. This requires all the threads to run the same program and the computational resources will be equally distributed to each thread. If one thread takes a much longer time to be completed, the overall performance will drop dramatically. The parallelism enables the GPU to be much more efficient than a CPU. However, this also increases the difficulty in programming. Generally, a GPU has less on-chip cache

and less flow control logic compared to the CPU. Therefore, the algorithm should have very few conditional branches.

- Limited inter-processor communication

Parallel computing often requires data sharing between different threads. However, due to the large memory latency, sharing data between threads through on-board memory is not efficient. Both Intel and the NVIDIA GPU have certain efficient ways of sharing data between limited threads through on-chip memory. If the algorithm requires all the threads to be tightly coupled and requires a large memory bandwidth between the inter-thread communications, the scalability of the algorithm implementation will be limited and the performance will be poor. To achieve the best performance, the algorithm should have methodologies to separate the tasks into a number of sub-tasks which require less communication between each other. Since Velodyne has 64 laser beams and provides more than 1 million points per sec, it is effective if the algorithm is operated over a single beam/point or over a single grid cell.

Figure 5-3 demonstrates the CUDA architecture. The function executed in the GPU is called a *kernel* or *grid*. Each kernel consists of a number of blocks and each block consists of a number of threads. Both the block index and the thread index can be one-, two- or three-dimensional.

The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks

Figure 5-3: Logical architecture of CUDA kernels

In this chapter, if a kernel is named `Kernel(.)`, the C code `Kernel <<< A,B >>> (input_data)` means that there are A number of device blocks and each block contains B threads. Altogether there are AxB threads running in the GPU. In addition, CUDA supports three-dimensional grids and blocks, so that the blocks can be launched in three-dimensional matrices, with indexes `blockIdx.x`, `blockIdx.y` and `blockIdx.z`. Similarly, the threads may have index `threadIdx.x`, `threadIdx.y` and `threadIdx.z` which would be used in the function code. However, only the first dimension (`blockIdx.x` and `threadIdx.x`) is frequently used in our work.

The memory in CUDA is also classified into 5 types:

Device global: dedicated graphical memory on the GPU. Slower but shared by all thread blocks

Host: normal memory for the CPU computation

Device shared: on-chip cache, fast. Shared by all threads in the same thread block.

Device register: fast but very limited. Accessed by a single thread itself.

Device Constant/Texture: Read-only memory on the GPU. Faster than the global memory.

Therefore, the algorithm should take the memory access into account.

- Memory bandwidth and latency

The bottleneck of the computing could be the computational capability of the processors; it could also be the memory bandwidth and the memory latency.

A GPU, in general, has much larger memory bandwidth than that of a CPU. However, a GPU often has less cache memory on chip, and the memory latency of the GPU memory could be higher. To reduce the memory latency, the memory accessed by each thread must be near to each other (coalescing). In this way, the processor would be able to read a whole block of memory at once, and distribute this block into many pieces of data to each thread, or vice versa. If the memories are always accessed in sequential read/write mode, the drawbacks of the memory latency can be reduced to a minimum.

For the multi-core CPU, since the computational power is not that high compared to the GPU, the memory latency can still be well handled by the increasing on-chip cache. However, the current GPU has achieved nearly 1TFLOPS (10^9 floating point operations), and if memory latency cannot be mitigated, the large bandwidth will be wasted and the overall performance will be poor. Proper memory coalescing should be taken into account since it can significantly improve the performance, e.g. 100 times faster [67].

- Data structure

Although a dynamic data structure is very convenient for software/algorithm development, it is not easy for hardware implementations. The hardware memory is always located in sequential and static addresses. The operating system must find empty memory slots for dynamic memory access. Very few programmers need to worry about the delay in normal CPU applications. This is generally not a big issue for CPU programs since the bottleneck of the CPU is often the computational speed

rather than the memory bandwidth. However, a GPU can access the static sequential memory much faster. Executing frequent dynamic memory operations like new and delete a memory block is not efficient. NVIDIA CUDA-enabled GPUs do not support any dynamic memory access so far. All the dynamic memory access must be handled by a CPU instead. This limitation indicates that all the dynamic data structure must be avoided, e.g. tree structure, linked array, etc. Even though it can be supported in the future, the dynamic data structure must be used with care, i.e., as little as possible. Frequent use of dynamic data access will downgrade the system performance. Therefore, in our system, all the GPU memory is operated in static data structures like 1D or 2D arrays.

5.3. *High Speed RBGAF-SLAM*

5.3.1. *Considerations*

To enable GPU processing, RBGAF-SLAM has to be converted into a highly parallel architecture. Each procedure of the RBGAF-SLAM update process is reformulated and separated into a large number of threads. Besides this, steps are taken to improve the execution efficiency.

First, the approach keeps inter-thread dependencies to a minimum to reduce the waste of memory bandwidth. Each thread processes a single particle/chromosome. If intensive communication cannot be avoided, the parallel threads are separated into two parallel processes. The first process stores the data for inter-thread communication into the memory buffer, and the second process retrieves the data from the buffer.

In addition, the memory types for each element are optimized according to the memory consumption and the frequency of access. The most frequently accessed variables are located in on-chip registers or shared memory. The data processed by multiple threads in parallel is stripped and organized into special structures to favour memory coalescing. A large number of threads are launched to hide the latency of global memory (on-card graphical memory).

For each function in RBGAF-SLAM, these two issues have to be solved in order to form highly parallel RBGAF-SLAM. The function has to be separated into many parallel threads and the memory used in the function has to be carefully re-mapped into proper memory format.

5.3.2. *Function design*

In GPU programming, the algorithm must be separated into thousands of threads to guarantee a highly paralleled architecture. The detailed description below shows how the algorithms are separated into this highly parallel architecture.

5.3.3. *Memory design*

For RBGAF-SLAM, the robot trajectories and the associated chromosome maps are the major memory consumptions. These occupy most of the on-card graphical memory. Practically, the number of particles is restricted by the memory space, especially for the 3D RBGAF-SLAM. By the definition of RBGAF-SLAM, the most natural data structure is a one-dimensional array and each element represents the i th ($i = 1 \sim P$) particle (trajectory&map pair), as shown in Figure 5-4.

Figure 5-4: Natural data structure for RB particle (trajectory and map pair)

However, with this naïve data structure, several problems will occur for GPU processing. In the parallel architecture, each thread will process a function for only one element, either for one robot pose or for one map chromosome. Therefore, the neighbouring threads will read an irregular size of memory that does not satisfy the coalescing requirement. To solve this problem, the memory element is unified into

a 32 bit floating number. For example, if there are three parameters to represent the robot pose (a, b and bearing), there will be three arrays with each array containing one parameter for all the poses in one trajectory. Therefore, when the threads are processing in parallel, memory coalescing can be guaranteed.

The other problem is that it is a very difficult task to add, delete or insert the map chromosome. This difficulty is due to memory conflicts. When multiple threads are processing different map chromosomes in the same map, the requirement to add, delete or insert an element may occur at the same time. In addition, when a map chromosome is deleted or inserted into the array, all the other map chromosomes are affected. Unless a dynamic data structure like a linked array is used, it is difficult to maintain the sequence of the map chromosomes.

Fortunately, in RBGAF-SLAM, the map chromosomes do not require any set sequence in memory allocation. Unlike the occupancy grid map, switching the values of two map chromosomes in the same map will not affect the map representation. This advantage can be used to simplify the operations.

Since there is a large number of map chromosomes, they are separated into many smaller groups (e.g. 100~300 groups for one map). Each group of map chromosomes will be processed by one thread at a time. In such a design, adding, deleting and inserting is the operation in one group so that all the threads are independent of each other. Each group records the number of elements inside the group. When a new chromosome is generated or inserted into the group, the number of elements increases by one and the data is added into the end of the array. When a map chromosome is to be deleted, the last element overwrites it, with the number of elements reduced by one.

5.3.4. *Map update*

The map update consists of several processes including map chromosome evaluation, chromosome selection and chromosome reproduction. The parallel architecture separates each RB particle (trajectory map pair) into one block and each map chromosome group into one thread. Therefore, there are P blocks and

each block contains 100~300 threads. The choice of the thread number 100~300 is due to the hardware specification. It requires at least 100 threads in the same block to hide graphic memory latency efficiently. However, if too many threads are designed, the shared memory and registers in each thread will be limited.

The map update of the j th map chromosome $m_t^{i,j}$ in the i th block is based on the robot pose x_t^i . The thread reads both $m_t^{i,j}$ and x_t^i for every processing.

First, each map chromosome is evaluated. Then a GPU kernel is launched as follows.

MapEvaluation

```
<<<TotalTrajectoryNumber, TotalGroupNumber>>>(map,pose,sensor_readings)
```

In this kernel, the relative range and bearing of the map chromosome and the robot pose are calculated. In respect of the sensor readings, the fitness value (weight) of the map chromosome is calculated and stored. If the map chromosome is ‘observed’ by a sensor reading, i.e., the distance from the chromosome position and the object detected by the sensor is less than the resolution threshold ε , the count value of the map chromosome will be increased by one.

Second, the map chromosomes are selected based on the fitness values (weight) obtained in the evaluation stage. The selection process reads two chromosomes in the group and compares the fitness values associated with the two chromosomes.

MapCompetition

```
<<< TotalTrajectoryNumber, TotalGroupNumber >>>(map,mapfitness)
```

The one with higher fitness values (weight) will survive and overwrite the failed one. However, a map chromosome which is outside the observation range will not be processed in the map chromosome selection.

Figure 5-5: CUDA kernels for map selection

As demonstrated in Figure 5-5, the map update process has TotalTrajectoryNumber multiplied by TotalGroupNumber threads running in parallel. All the threads are independent of each other without memory conflicts.

5.3.5. Generating new map chromosomes

In the map update process, if the chromosome is observed, the corresponding sensor reading is verified and marked. For the marked readings, the count value of the corresponding map chromosome is increased by one in the map evaluation stage. This operation performs the chromosome compression.

On the other hand, sensor readings which are not marked indicate new objects and new chromosomes should be created for them. A new chromosome is generated for every new unverified sensor reading. However, the sensor readings are filtered by a minimum range threshold and a maximum range threshold.

For a highly parallel architecture, the number of threads in each block can be set to the number of sensor readings per scan. For ease of generating map chromosomes without memory conflicts, the number of threads is set to be the same as the group number, or, alternatively, the group number is set to be a factor of the number of readings per scan. For example, if there are 360 readings per scan, the group number can be chosen as 120 or 180. The function for generating new map chromosomes is as follows in Figure 5-6:

MapGenerate

<<<TotalTrajectoryNumber, TotalGroupNumber>>>(map,sensor_readings)

Figure 5-6: CUDA kernels for map chromosome generation

5.3.6. *RB-Particles (robot trajectory & map pair) evaluation*

To evaluate the RB-Particle (robot trajectory & map pair), there are three stages to calculate the weights. The first stage is to evaluate the weight providing a map chromosome and a robot pose. The second stage is to evaluate the weight providing a map consisting of a large number of chromosomes and a robot pose. The last stage is to evaluate the weight (likelihood) providing a map and a robot trajectory which consists of a large number of robot poses. Figure 5-7 illustrates the three stages of the process

Figure 5-7: CUDA kernels for RB-Particle evaluation

For the independence of the threads, the intermediate results are stored in the GPU memory in array/matrix formats. A ChromosomePoseWeight is associated with every chromosome. Therefore, it is a 3D matrix of TotalTrajectoryNumber by TotalChromosomeNumber by TotalDepth. The MapPoseWeight is a 2D matrix of TotalTrajectoryNumber by TotalDepth. The RB-Particle weight is then a 1D array, ready for the re-sampling process.

5.3.7. *Final resampling*

With the 1D array which contains the RB-Particle weights, the competition selection process is easy to implement. However, to copy the RB-Particle (robot trajectory & map pair) is computational costly. In a highly parallel architecture, this operation is completed by thousands of threads and each thread only copies one

element at a time. The process has three stages. The first stage is to implement the competition selection and determine the winners. The results are stored in one 1D array. The second stage is to copy the winner trajectory and overwrite the loser, based on the results stored. The third stage is to copy the map chromosomes based on the results stored. This process finalizes the whole updating process, as shown in Figure 5-8.

Figure 5-8: CUDA kernels for final resampling

5.4. Experimental results

Mobile robot navigation requires real-time processing for SLAM. A solution which provides accurate results may not be practical unless it can be processed in real time. The efficiency of the algorithm is also related to the implementation. If the algorithm is easy to implement in hardware with higher efficiency, it is better. The hardware processing efficiency is different for different algorithms. Therefore, if the hardware efficiency is taken into consideration in the algorithm development, it will further improve the overall processing efficiency.

Besides, the limitation of CPU processing is that the CPU contains lots of functions which are not used in the calculations. If the mobile robot navigation algorithm can be processed by the GPU, which has much higher GFLOPS ratings, real-time processing will be easier to achieve. However, the algorithm must be designed in such a way that it can be processed in a highly parallel architecture for GPU

processing. RBGAF-SLAM is one kind. The updating times of the SLAM approaches are shown in Table 5-2. The updating time of DP-SLAM (CPU processing), RBGAF-SLAM (CPU processing) and the RBGAF-SLAM (GPU processing) are compared. They are tested with the same number of Rao-Blackwellized particles.

TABLE 5-2: PROCESSING TIME OF SLAM APPROACHES

	DP-SLAM (CPU)	RBGAF-SLAM (CPU)	RBGAF-SLAM (GPU)
Low resolution	2.3 s	2.1 s	0.12 s
High resolution	not real-time	8.2 s	0.47 s

Although the requirement for real-time processing varies in different applications, it can be generally considered that 5 Hz is an acceptable frequency for mobile robot SLAM. Or, in other words, if the updating time for the SLAM algorithm is less than 200 ms, it can be considered as real time. If the updating time is more than 1 s, we may call it ‘marginal real time’. Table 5-1 shows that GPU processing speed is much faster than the processing speed of a CPU. The highly parallel ultra speed RBGAF-SLAM is a real-time approach, while the DP-SLAM is marginal real time as described in [13]. In our implementations, the ultra speed RBGAF-SLAM can process more than 1,000 particles (trajectory/map pair) with acceptable speed. It also has the potential to process more if better optimization and multi-GPU ‘SLI’ mode [67] can be implemented.

Compared to the DP-SLAM, the achievement of real-time 3D SLAM is one of the outstanding advantages of RBGAF-SLAM.

5.5. *Summary*

To achieve real-time processing of the RBGAF-SLAM, a highly parallel architecture is needed. A graphical processing unit (GPU) is chosen for highly parallel RBGAF-SLAM due to its significant advantage in computational speed. RBGAF-SLAM is modified to adapt to the highly parallel GPU processing. It is separated into a sequence of operations which can be processed by a large number of independent parallel threads. Memory conflicts are avoided to guarantee stable parallel computing. Memory coalescing is also taken into consideration to achieve the best memory bandwidth. Dynamic data structure is avoided for the GPU implementation. Complex function flows are also avoided in the highly parallel architecture. With these improvements, RBGAF-SLAM can be processed by a SIMD GPU. Significant speed boosts are shown in our GPU implementations and real-time processing speeds are achieved.

CHAPTER V: DISCUSSION AND CONCLUSIONS

6.1. Discussion

Localization is one of the fundamental requirements for the mobile robot to complete the task. With the odometry/INS system only, accuracy is not sufficient to complete most of the tasks. If an environmental map is not available, the robot has to localize itself and to build the environmental map simultaneously. Simultaneous localization and mapping (SLAM) addresses this issue within a unified framework. If prior information about the environment is not available or highly unstructured non-feature based SLAM approaches or view based approaches are most suitable.

The occupancy grid map is a popular map representation for non-feature based or view based SLAM. It is a nonparametric method which can represent the environment with raw sensor measurements. Particle filters, instead, are the nonparametric method frequently used to estimate the state of a dynamic system. If particle filters are used for map representation, the advantages are threefold. Firstly, the resampling process will reduce the number of particles needed, mitigating the sample depletion problem and result in a more efficient map representation. Secondly, the computational cost increases according to the number of objects observed but not the size of the space navigated. The approach strips out ‘not-important’ areas like ‘empty space’ and ‘unknown space’ from the environmental map so that both computational time and memory consumption can be reduced with a negligible penalty in the positioning accuracy. The third important advantage is that, unlike occupancy grid cells, map particles can be stored in any sequence and can be easily separated, combined, interchanged or deleted. This advantage greatly eases the hardware implementation of a particle map.

However, to apply particle filters to map updating is difficult. The sample impoverishment problem caused by Monte Carlo resampling will discard the map

particles. The robot movement also requires convenient ways of adding new map particles. By viewing the map update as an optimization process, the proposed genetic algorithmic filter (GAF) combined with a compressed chromosome map for environmental map representation effectively address these critical research issues. It is much easier and very much more reliable to balance the divergence and convergence with GA techniques. In simulations and experiments, they prove to work successfully over compressed chromosome set representing the environmental map. With obstacles of any shapes, sizes, and densities, GA can update and optimize the map without discarding it. Genetic algorithmic filter is much better for map update than particle filters with Monte Carlo resampling. The compressed chromosome map is also a better map representation compared to the naïve particle map.

Besides, the GAF mapping still benefits from the advantages of particle filters. Simulations and experimental results show that compared to occupancy grid mapping, GAF mapping require much less computational resources under the same map resolution. This advantage is very significant in 3D space, or at high resolution (relative to the sensor accuracy). However, in low resolution (relative to the sensor accuracy) 2D mapping, the advantage of GAF mapping is less significant. Besides, GAF mapping does not map the ‘unknown space’ and the ‘empty space’. This will not affect SLAM results, but such information might be useful in other applications like path planning.

Researches and implementations have proven that a Rao-Blackwellized particle filter (RBPF) approach is an effective means to estimate the full SLAM posterior. The proposed Rao-Blackwellized Genetic Algorithmic Filter SLAM which combined RBPF with genetic algorithmic filter performs effectively for solving the view based approach to SLAM. The simulation and experimental results show the RBGAF-SLAM performances with different sensors, in different environments, both 2D and 3D, and on different robot platforms. The localization error is bounded by the diversity of the Rao-Blackwellized particles (trajectory/map pair), indicating consistent SLAM performance.

Compared to the feature based RBPF-SLAM approaches, better performance of RBGAF-SLAM can be expected when the prior environmental knowledge is not available and reliable feature extraction is difficult. However, when the environment can be ‘predicted’, especially in structured environment, sufficient reliable features can be detected. In this case, feature based approaches will be more efficient.

Compared to occupancy grid map based RBPF-SLAM, like DP-SLAM, RBGAF-SLAM is more efficient in most of the navigation scenarios, especially in 3D environment. With the same computational resources, higher efficiency will give better performance because more RB particles can be processed. However, for SLAM with low-resolution 2D maps, the advantage is less significant.

As presented in [13], DP-SLAM is considered to be near real-time. RBGAF-SLAM, instead, thanks to the advantages of compressed chromosome representation, can be ‘split’ into a highly parallel architecture, executing in a great number of independent threads. Therefore, with a Graphic Processing Unit (GPU), RBGAF-SLAM achieves a huge speed boost-up compared to normal CPU implementations. Thus, a real time speed can be assured. The highly parallel architecture of RBGAF-SLAM yields an outstanding advantage because it achieves real time speed which is difficult for DP-SLAM.

6.2. Conclusions

The genetic algorithmic filter and compressed chromosome map provide a suitable way of representing a non-parametric environmental map with raw sensor measurements. The use of raw sensor measurements directly obviates the need for complex feature extraction. At the same time, the genetic algorithmic filter still facilitates the use of any arbitrary motion estimation models or sensor models. It is more computationally efficient than the popular occupancy grid mapping.

Rao-Blackwellized Genetic Algorithmic Filter SLAM (RBGAF-SLAM) proposed for mobile robot navigation is an efficient SLAM algorithm based on Rao-Blackwellized particle filters and a compressed chromosome map for recovering the full SLAM posterior. RBGAF-SLAM is more computationally efficient than DP-SLAM in most navigation scenarios. Besides, it is more reliable than feature based SLAM without prior knowledge in highly complicated, un-structured environment.

RBGAF-SLAM in a highly parallel architecture is implemented on a Graphic Processing Unit (GPU). We have shown that it can achieve real time processing speed on a market GPU. This outstanding advantage can greatly save the space and weight required for computing devices to achieve real time speed.

In a summary, the specific research contributions of this thesis made to the problem of simultaneous localization and mapping (SLAM) may be summarized as follows:

1. Development of an efficient and flexible genetic algorithmic map representation for view-based SLAM approaches. It avoids sample impoverishment problem of Monte Carlo resampling and is more efficient than occupancy grid map, especially appropriate in 3D environment.
2. Synthesis of an efficient real-time view-based SLAM approach: the Rao-Blackwellized Genetic Algorithmic Filter SLAM. Such SLAM approach does not rely on feature extraction, and is capable of integrating arbitrary sensor and motion models.
3. Development of a highly efficient parallel computing structure for real-time RBGAF-SLAM. The approach can be implemented on GPU which has much more computational resources and this provides significant improvement in processing speed SLAM. As a result, real time 3D view based SLAM can be achieved.

6.3. *Suggestions for future research*

With the results of the investigations in this thesis, several possible avenues of future research effort can be proposed.

Each SLAM algorithm has its advantages and disadvantages. Therefore, it is wise to use a specific SLAM algorithm in specific scenarios. For example, if there is prior knowledge available and feature extraction can be reliably performed, feature-based approaches would be a good choice. With reliable features, the computational cost required for SLAM is much less, and hence it is more efficient. However, ways of combining RBGAF-SLAM with other SLAM approaches have not been investigated. How can the SLAM algorithms, including RBGAF-SLAM, be chosen automatically during navigation? Both DP-SLAM and RBGAF-SLAM use non-parametric methods for map representation. Is it possible to convert a compressed chromosome map into an occupancy grid map, or vice versa? How to switch SLAM algorithms in real time when necessary? These are still open questions and needs further investigations.

Finally, due to hardware limitations, implementations of RBGAF-SLAM cannot cover all types of sensors and all robot platforms. We believe that it can perform well with other sensors, but it is worth investigating these topics, especially for sensors which have sophisticated sensor models.

PUBLICATIONS

- J.F. Dong, S. Wijesoma, and A. P. Shacklock, “An Efficient Rao-Blackwellized Genetic Algorithmic Filter for SLAM”, in *IEEE International Conference on Robotics and Automation*, Italy, 2007.
- J.F. Dong, S. Wijesoma, and A. P. Shacklock, “Extended Rao-Blackwellized genetic algorithmic filter SLAM in dynamic environment with raw sensor measurement”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.
- J.F. Dong, and S. Wijesoma, “Improving Rao-Blackwellized Genetic Algorithmic Filter SLAM Through Genetic Learning”, in *International Conference on Control, Automation, Robotics and Vision*, Hanoi, Vietnam, 2008.
- P. Moghadam, W.S. Wijesoma, and J.F. Dong, “Improving Path Planning and Mapping Based on Stereo Vision and Lidar”, in *International Conference on Control, Automation, Robotics and Vision*, Hanoi, Vietnam, 2008.

REFERENCES

- [1] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A factored solution to the simultaneous localization and mapping problem”, *Association for the Advancement of Artificial Intelligence (AAAI)*, Edmonton, Canada, 2002.
- [2] W. Rencken, “Concurrent Localization and Map Building for Mobile Robots Using Ultrasonic Sensors”, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Yokohama, Japan, 1993.
- [3] J. Aulinas, Y. Petillot, J. Salvi and X. Lladó, “The SLAM problem: a survey”, in *International Congress of the Catalan Association of Artificial Intelligence*, 2008.
- [4] J. Borenstein, “Internal Correction of Dead-reckoning Errors With the Compliant Linkage Vehicle”, *Journal of Robotic Systems*, 12(4), pp. 257-273, April 1995.
- [5] B. Barshan, and D. Whyte, “Inertial Navigation Systems Mobile Robots”, *IEEE Transactions on Robotics and Automation*, 11(3), pp. 328-342, June 1995.
- [6] I. Getting, “The Global Positioning System”, *IEEE Spectrum*, pp. 36-47, December 1993.
- [7] B. Azimi-Sadjadi and P.S. Krishnaprasad, “Integer Ambiguity Resolution in GPS using Particle Filtering”, in *Proceedings of the American Control Conference*, Arlington, June 2001.
- [8] A. Doucet, N. de Freitas, K. Murphy, and S. Russel, “Rao-Blackwellized particle filtering for dynamic Bayesian networks”, in *Proceeding of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.
- [9] Dirk Hahnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun, “An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environment from Raw Laser Range Measurements”, in *International Conference on Intelligent Robots and Systems*, 1, pp. 206-211, 2003,.

- [10] J. Minguez, F. Lamiraux, and L. Montesano, "Metric-Based Scan Matching Algorithms for Mobile Robot Displacement Estimation", in *IEEE International Conference on Robotics and Automation*, 2005.
- [11] Sebastian Thrun, Dieter Fox, and Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 221-242, 2004,.
- [12] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellised particle filters by adaptive proposals and selective resampling", in *International Conference on Robotics and Automation*, Barcelona, Spain, 2005.
- [13] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks", in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.
- [14] Sebastian Thrun, Dieter Fox, Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 77-85, 2004.
- [15] U. Nehmzow and C. Owen, "Robot navigation in the real world: Experiments with Manchester's FortyTwo in unmodified, large environments", *Robotics and Autonomous Systems*, 33(4), pp. 223-242, 2000.
- [16] Y. Z. Peng, K. Yuan, and J. C. Liu, "A three-layer planning navigation method for in door mobile robot", *Electric Machines and Control*, 10(4), pp. 380-384, 2006.
- [17] Jonathan Courbon, Youcef Mezouar, Laurent Eck and Philippe Martinet, "A generic framework for topological navigation of urban vehicle", in *International workshop on Safe navigation in open and dynamic environments Application to autonomous vehicles*, Kobe, Japan, 2009.
- [18] Y. Zhuang, S. D. Xu, and W. Wang, "Mobile robot geometric-topological map building and self-localization", *Control and Decision*, 20 (7), pp. 815-818, 2005.
- [19] J. Borenstein, H.R. Everett, L. Feng, and D. Wehe, "Mobile Robot Positioning- Sensors and Techniques", *Journal of Robotic Systems, Special Issue on Mobile Robots*. 14(4), pp. 231 – 249, 1997.
- [20] Sebastian Thrun, Dieter Fox, and Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 187-218, 2004.

- [21] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems", in *Proceedings of AeroSense*, 1997.
- [22] E. Wan and R. van der Merwe, "The Unscented Kalman Filter", *Kalman Filtering and Neural Networks*, Chapter 7: Wiley, 2001.
- [23] S. Thrun, "Robotic mapping: A survey", *Exploring Artificial Intelligence in the New Millenium*, Morgan Kaufmann, San Francisco, 2002.
- [24] Sebastian Thrun, Dieter Fox, and Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 157-184, 2004.
- [25] Sebastian Thrun, Dieter Fox, and Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 121-154, 2004.
- [26] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map building algorithm for real time implementation", *Trans. of Robotics and Automation*, 2001.
- [27] G. Dissanayake, P. Newman, et al., "A solution to the simultaneous localisation and map building (SLAM) problem", *IEEE Trans. Robotics and Automation*, 2001.
- [28] J. Guivant, "Efficient Simultaneous Localization and Mapping in Large Environments", *PhD thesis*, The University of Sydney, 2002.
- [29] M. Montemerlo, S. Thrun, D. Koller and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges", in *International Joint Conferences on Artificial Intelligence*, 2003.
- [30] Jonathan Courbon, Youcef Mezouar, and Philippe Martinet, "Autonomous Navigation of Vehicles from a Visual Memory Using a Generic Camera Model", *IEEE Transactions on intelligent transportation systems*, 10(3), pp. 392-403 2009.
- [31] Hong Zhang, Bo Li and Dan Yang, "Keyframe Detection for Appearance-Based Visual SLAM", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, October 2010,
- [32] Cooper Bills, Joyce Chen, and Ashutosh Saxena, "Autonomous MAV Flight in Indoor Environments using Single Image Perspective Cues", in *International Conference on Robotics and Automation (ICRA)*, 2011.

- [33] J Jonathan Courbon, Youcef Mezouar, Nicolas Guenard and Philippe Martinet, “Vision-based navigation of unmanned aerial vehicles”, *Control Engineering Practice*, 18, pp 789–799, 2010.
- [34] Jonathan Courbon, Youcef Mezouar, Nicolas Guenard and Philippe Martinet, “Visual navigation of a quadrotor aerial vehicle”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis USA, 2009.
- [35] Josep Aulinas, “Msc Thesis dissertation: 3D Visual SLAM applied to large-scale underwater scenarios”, Institute of Informatics and Applications, University of Girona, Girona, 2008.
- [36] José Neira, “Pure Visual SLAM”, in *ICRA workshop on visual navigation*, 2008.
- [37] D. Lowe, “Object recognition from local scale-invariant features”, in *Proceedings of the International Conference on Computer Vision*, 2, pp. 1150–1157, 1999.
- [38] Andrea Garulli, Antonio Giannitrapani, Andrea Rossi, and Antonio Vicino, “Mobile robot SLAM for line-based environment representation”, in *44th IEEE Conference on Decision and Control, and the European Control Conference*, Seville, Spain, December 2005.
- [39] L. H. Gao, F. Fang, and S. D. Ma, “A sonar-based map building approach for mobile robot”, *Manufacturing Automation*, 28(11), pp. 33-35, 2006.
- [40] J. M. Carmena and J. C. T. Hallam, “The use of Doppler in sonar-based mobile robot navigation: inspirations from biology”, *Information Sciences*, 161(1-2), pp. 71-94, 2004.
- [41] R. Carelli and E. O. Freire, “Corridor navigation and wall-following stable control for sonar-based mobile robots”, *Robotics and Autonomous Systems*, 45(3-4), pp. 235- 247, 2003.
- [42] Franz Andert and Lukas Goormann, “Combined Grid and Feature-Based Occupancy Map Building in Large Outdoor Environments”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, USA, 2007.

- [43] Sebastian Thrun, Dieter Fox, and Wolfram Burgard, *Probabilistic Robotics*, Summer School, pp. 221-242, 2004.
- [44] Y. Duan and X. H. Xu, "Navigation for mobile robot based on uncertainty grid-map", *Control Theory & Applications*, 23(6), pp. 1009-1013, 2006.
- [45] R. H. Luo, B. R. Hong, and M. H. Li, "Grid map building based on prediction of local features", *Journal of Harbin Institute of Technology*, 36 (7), pp. 877-879, 2004.
- [46] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods In Practice*, Springer, 2001.
- [47] J. Boluda and J. Domingo, "On the advantages of combining differential algorithms and log-polar vision for detection of self-motion from a mobile robot", *Robotics and Autonomous Systems*, 37(4), pp. 283-296, 2001.
- [48] J. J. Guerrero and C. Sagues, "Uncalibrated vision based on lines for robot navigation", *Mechatronics*, 11(2), pp. 759-777, 2001.
- [49] N. de Freitas, M. Niranjan, A. Gee, and A. Doucet, "Sequential monte carlo methods to train neural network models", *Neural Computation*, 12(4), 2000.
- [50] K. Kanazawa, D. Koller, and S.J. Russell, "Stochastic simulation algorithms for dynamic probabilistic networks", in *Conference on Uncertainty in Artificial Intelligence*, 1995.
- [51] F. Evennou, F. Marx, and E. Novakov, "Map-aided indoor mobile positioning system using particle filter", in *Conference on Wireless Communications and Networking*, 4, pp. 2490-2494, March 2005.
- [52] B. Barshan and H. Durrant-Whyte, "An Inertial Navigation System for a Mobile Robot", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Yokohama, Japan, pp. 2243-2248, July 1993.
- [53] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Massachusetts, 1989.
- [54] GPS Report, Potomac, MD: *Phillips Business Information*, November 1992.
- [55] Player project, Internet:<http://playerstage.sourceforge.net/>, 2009.

- [56] James Kramer and Matthias Scheutz, "Development Environments for Autonomous Mobile Robots: A Survey", *Autonomous Robots*, 22(2), pp. 101-132, 2007.
- [57] Orca project, Internet: <http://orca-robotics.sourceforge.net/>, 2008.
- [58] J.F. Dong, S. Wijesoma, and A. P. Shacklock, "An Efficient Rao-Blackwellized Genetic Algorithmic Filter for SLAM", in *IEEE International Conference on Robotics and Automation*, Italy, 2007.
- [59] H. Rong, H. Binghong, "Coevolutionary Particle filter for Simultaneously localization and mapping", in *Proceeding of NLP-KE*, 2005.
- [60] N. M. Kwok, Gu Fang and Weizhen Zhou, "Evolutionary Particle Filter: Re-Sampling from the Genetic Algorithm Perspective", in *IEEE International Conference on Intelligent Robots and Systems*, 2005.
- [61] J. Bohren, T. Foote, J.Keller, et al., "Little Ben: The Ben Franklin Racing Team's Entry in the 2007 DARPA Urban Challenge", *Journal of Field Robotics*, 25(9), 598-614, 2008.
- [62] S. Kammel, J. Ziegler, B. Pitzer, et al., "Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge", *Journal of Field Robotics*, 25(9), pp. 615-639, 2008.
- [63] Dmitri Dolgov and Sebastian Thrun, "Detection of Principal Directions in Unknown Environments for Autonomous Navigation", in *Proceedings of the Robotics: Science and Systems IV*, June 2008.
- [64] Ee Hui Lim and David Suter, "Multi-scale Conditional Random Fields for Over-segmented Irregular 3D Point Clouds Classification", *Computer Vision and Pattern Recognition Workshops*, 2008.
- [65] Austin I. Eliazar and Ronald Parr, "DP-SLAM 2.0", in *IEEE International Conference on Robotics and Automation*, 2004.
- [66] J.F. Dong, S. Wijesoma, and A. P. Shacklock, "Extended Rao-Blackwellised genetic algorithmic filter SLAM in dynamic environment with raw sensor measurement", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.
- [67] Nvidia Co., "CUDA programming Guide2.3", Internet: http://developer.nvidia.com/object/cuda_3_2_downloads.html, 2010.

- [68] Jun-Sik Kim, Myung Hwangbo, Takeo Kanade, “Realtime Affine-photometric KLT Feature Tracker on GPU in CUDA Framework”, in *Workshop on embedded Computer Vision*, October 2009.
- [69] L. Seiler, “Larrabee: A Many-Core x86 Architecture for Visual Computing”, *ACM Transactions on Graphics*, 27(3), Article 18, August 2008.
- [70] Gordon Petrie, “An Introduction to the Technology- Mobile Mapping Systems”, *GEO informatics*, pp32-43, 2010.
- [71] M. Himmelsbach, A. Muller, T Luttel and H. J. Wunsche, “LIDAR-based 3D Object Perception”, in *Proceedings of 1st International Workshop on Cognition for Technical. Systems*, Munich, Germany, October 2008.
- [72] M. Montemerlo, “The DARPA Urban Challenge: Perspectives on Urban Autonomous Driving”, Stanford Artificial Intelligence Lab, NIPS, 2007.
- [73] F. Moosmann, O. Pink and C. Stiller, “Segmentation of 3D Lidar Data I non-flat Urban Environment using a Local Convexity Criterion”, in *Intelligent Vehicles Symposium*, 2009.
- [74] D. Steinhauser, O. Ruepp, and D. Burschka, “Motion Segmentation and Scene Classification from 3D LIDAR Data”, in *IEEE Intelligent Vehicles Symposium*, Netherlands, June 2008.
- [75] Nikolay Chumerin and Marc M. Van Hulle, “Ground Plane Estimation Based on Dense Stereo Disparity”, in *International conference on Neural Networks and Artificial Intelligence*, 2008.
- [76] John Leonard, Jonathan How, Seth Teller, et al., “A Perception-Driven Autonomous Urban Vehicle”, *Journal of Field Robotics*, 25(10), pp. 727–774, 2008.

APPENDIX A: VELODYNE 3D LIDAR PROCESSING ALGORITHMS

A.1. Velodyne processing algorithms

The dense measurement data from the surrounding environment gives sufficient information for mobile robot navigation. Obstacle detection, environmental mapping, target tracking, classification and path planning algorithms all benefit from the dense 3D range and bearing measurements.

Obstacle detection is commonly processed in the vertical direction and the results are significant to the success of environmental map as in [70]. The height values of each laser point are calibrated and examined to classify the ground vs. obstacles. Similar approaches have been adopted by researchers [71] due to their efficiency and low computational cost. However, the inter-beam errors and the environmental noises make the results unreliable. To mitigate these drawbacks, per beam edge detection is also processed. This approach is efficient and can be used to detect small obstacles. In DARPA Urban teams [72], laser point density, laser beam density and the height differences are combined to detect the obstacles over the ground plane. However, while the obstacles can be identified by these approaches, the ground inclination angles are not examined in detail. Although more sophisticated obstacle/ground detection algorithms have been proposed, the current CPU speed cannot guarantee to process them in real time. Therefore, researchers have to take computational cost into account. The current real-time algorithms for the Velodyne obstacle detection are all ‘simple’ though computationally efficient.

Besides obstacle detection, the Velodyne laser range finder can do other tasks. In [71], the model-based classification algorithms are developed for the 3D laser range finders to identify cars. The classification is based on the predefined point cloud model and is claimed to achieve real-time speed. However, only one class is identified. Sophisticated model-based approaches are very useful for the classification, but more

classes would involve more computational cost and the real-time performance may not be guaranteed. In addition, more experiments are also needed, as described in [71]. In [73], the classification is done based on a local convexity measures. The results look impressive but the processing speed is not in real time. More robust obstacle detection approaches are needed.

The Velodyne can also be used for high resolution mapping, providing camera-like views in the infra-red spectrum and detecting road lanes [61]. In [64], a visual-odometry-like approach is proposed to correct the vehicle odometry. The Velodyne information is also used for a target tracking algorithm by comparing the differences between different frames. The approach can detect moving objects and filter out the static environment. The moving targets are then detected and tracked by particle filters. There would be a minimum target speed for a successful target tracking. Especially in a crowded environment, the computational cost to track all the targets of different classes would be a challenge due to the limited computational resources.

Other researchers are developing Velodyne processing algorithms with less concern about computational speed. In [73], Velodyne data is processed in the camera-view for segmentation and classification, but the processing speed cannot reach the real-time requirement. In [74], the Velodyne frame data is processed to provide feedback information to correct the vehicle odometry readings. The corrected vehicle pose information is then used to detect the moving targets. In [71], the model-based target tracking algorithm can run in real time, but it is only used for single target tracking. If more targets are to be classified and tracked, it would be a challenge to achieve the real-time requirement again.

The current real-time approaches for the Velodyne processing are restricted to ‘efficient’ algorithms. Obstacle detection in a structured urban environment is achieved in real time. Other tasks, like multi-class classification and multi-target tracking, especially in an unstructured environment, are still a challenge. However, the limitation is not in the algorithms, but the computational resources. If redundant computational resources can be provided, we believe that the challenge could be solved. Although CPU speed is improving fast, the 3D LiDAR is also improving fast.

The data rate has increased from 1 M/s to 2.2 M/s, requiring more computational resources. Due to the space and weight limitation on a mobile robot platform, the computational resources are limited. Software algorithms need to take the hardware limitations into account. A solution is needed for processing sophisticated algorithms in real time.

A.2. Our Velodyne processing algorithms

This section will introduce the Velodyne perception algorithms used in our system, followed by the GPU implementation of these algorithms.

Figure 7-1: Overall Processing System Block Diagram

As shown in Figure 7-1, the raw Velodyne UDP data packets are captured, interpreted and calibrated with the vehicle's pose data to obtain the 3D points in Cartesian coordinates. These point clouds are then filtered to remove any spurious noise, including outliers. The filtered point cloud data is then processed in different modules. The laser points are first classified into one of two classes, i.e. obstacle or ground, by the ground/obstacle classifier. The output of this binary classifier can be fed into the RBGAF-SLAM module. The curb detection module segments the curb points and produces a curb map. This helps to estimate the road boundaries without vision [75]. The surface or slope detection module determines possible planar surface patches and

their inclination angles. In the following section, each of these function blocks is described in detail along with the specific processing algorithms.

A.3. Pre-Filtering

The Velodyne sensor data often contains spurious data due to dust and spurious reflections. Hence pre-filtering is necessary before other operations are applied to the data. We use a median filter in the pre-filter to remove outliers.

A.4. Ground/obstacle classification

Ground or traversable terrain detection is challenging since traversable terrain may involve slopes of small gradient. If they are not properly detected, it may be misinterpreted as an obstacle by the laser system. Further, the traversable terrain may not be perfectly flat, especially in cross-country environments. The problem is exacerbated due to the strong vibrations in the vehicle to which the sensor is subjected when the engine is on or the vehicle is moving, especially at higher speeds. All of the above factors, together with the sensor's systematic and random noise, make the problem formidable. It is of interest to note that although for each individual laser the range error may be small, the collective error of the system of laser beams is quite high. For example, we have found that the laser range measurement between different laser beams may have 10~30 cm errors, even after the calibration data is applied. This complicates ground detection and separation of small objects/obstacles (curbs) from ground.

To separate ground from obstacles, there are two considerations. The first is the obstacle/ground classification, to reliably detect dynamic objects such as moving cars and pedestrians, and static objects such as buildings, lamp-posts, trees, etc. The reliability of detecting these static and dynamic objects that constitute obstacles to the vehicle is of paramount importance, i.e. false negatives must be a minimum. The second is to detect slopes and reliably obtain their inclination angles to determine whether the surface is traversable or non-traversable, based on the vehicle's design. In [61] and [62], obstacle detection mainly focuses on the first consideration. The obstacles are detected but the ground inclination angles are not examined or are simply

approximated by radiation line segments. A surface might be very smooth but exceeds the maximum inclination that a vehicle can drive through. To avoid such misclassifications, both considerations must be examined at the same time.

A.5. Obstacle detection

The obstacle/ground classification is done via two approaches, as shown in Figure 7-2. The first is to detect small obstacles and compare the predicted beam pattern (profile) of each of the single beams incident on the ground surface and compare this with the actual data received to determine the discrepancies and hence obstacles through a gradient change algorithm. To detect large objects we compute a beam histogram over a local grid map and compare with the expected beam histogram to detect the obstacles. To improve robustness and accuracy, results from both approaches are fused to segment the point clouds into obstacles and ground surface.

Figure 7-2: The overall block diagram of the obstacle/ground classification module.

These two approaches have the same advantages. Firstly, they are less affected by environmental slopes and noises. Secondly, they are all per point processing, i.e., the operations are processing each laser point and independent of each other. As a result, the processing can be highly parallelized into thousands of threads and be quite flexible in the number of threads. Unlike model-based obstacle detection algorithms, these two approaches can be easily implemented in a GPU.

A.6. Obstacle detection

The experiments are conducted with a Velodyne laser range finder mounted on top of a vehicle, driving through urban and off-road environments. The localization data is obtained from a GPS with an IMU. The system provides very reliable ground/obstacle classification output in graphical user interface (GUI) as shown in Figure 7-3. The green points correspond to ground or traversable terrain whilst red points correspond to obstacles at varying distances from the Velodyne.

Figure 7-3: GUI view of obstacle detection result using 3D Velodyne data.

Red: obstacles. Green: ground.

A.7. Surface/Slope Detection

Figure 7-4 shows the diagram for surface detection. The laser points are recorded as $p(x, y, z)$ in the 3D space. If the laser points are projected onto a 2D plane (vehicle ground plane), then they can be represented as $p(x, y)$, with the altitude (z coordinate) of the points ignored, as in the curb line fitting.

Figure 7-4: Surface/Slope Estimation

To detect the surface norm and the inclination angle, in the polar coordinates, for every 30 degrees and every 4 meter depth, the environment is segmented into a number of 2D patches. All the laser points in the same patch are recorded in one list and processed to find the surface norm as well as the inclination angle of the surface on that patch. We use a RANSAC approach to detect and fit planes as follows.

1. All points on a selected 2D patch are organized into one list.
2. Three points are randomly selected from the list, $p1(a1,b1,c1)$ $p2(a2,b2,c2)$ and $p3(a3,b3,c3)$. Compute two vectors, $v21(a2-a1,b2-b1,c2-c1)$ and $v31(a3-a1,b3-b1,c3-c1)$. Calculate the normal vector by the cross product

$N = v21 \times v31$. This represents the normal to the surface containing the points, p1, p2 and p3. Now calculate the unit normal vector

$$n = \frac{N}{|N|}$$

For all the other laser points $p(a,b,c)$ in the same patch, calculate the distance to the surface by

$$d = |n \bullet p|$$

If d is less than a pre-selected threshold ($=25$ cm), the weight (vote) of this surface constituting the three points is increased by 1.

Repeat the process from 2 to 3, N ($=20$) times with three different sets of points selected randomly from the list. Now, from among the N different unit vectors, choose the one with the highest vote or weight as the best fit surface containing the list of points on the 2D patch. If the set weight is less than 70% of the total number of points T in that patch, it is not considered as a surface. If the unit normal vector is $n(x, y, z)$,

then the surface normal is $\alpha = \arctan\left(\frac{|z|}{\sqrt{x^2 + y^2}}\right)$ and the inclination angle will be $90 - \alpha$ (in degrees).

The above procedure is repeated for all patches on the 2D environment.

With the same system, the surface detection process is also taken at the same time as the obstacle detection. The red colour shows vertical surfaces and the green colour shows the horizontal surfaces. Figure 7-5 shows the results of surface/slope detection.

Figure 7-5: GUI view of surface/slope detection results.

Red: vertical surface. Green: flat surface. Grey: unclassified surface.

A.8. Curb detection and marking

To identify the traversable area, ground and obstacle binary classification is not enough [76]. Especially on urban roads it is necessary to detect the curbs alongside the road edges. The curb is too small in height to be detected using the approaches described for ground/obstacle classification above. Range edges are detected over every single laser beam. Since the inter-beam calibration contains larger errors, the processing is mainly done for individual beams. The results are then mapped to locate the potential curb points. These points are extended and connected for ease of visualization. Line fitting is the last step to approximate the curb lines at the road edges.

To detect the curbs, the data stream of each laser beam when swept through 360^0 is considered separately. If the ground is ideally flat and smooth, all the laser points from the same laser beam will trace a circle. However, at curb edges a distinct ‘Z’ shaped pattern would be produced.

For a single laser beam $z(i,j)$ from i^{th} beam in the j^{th} laser point of the data stream corresponding to one complete rotation, we take the first derivative,

$$\frac{\partial z(i,j)}{\partial j} = z(i,j) - z(i,j-1)$$

The second derivative is then calculated in a similar manner and threshold operations are processed over the second derivative to determine the potential curb points.

The cells representing potential curb points are extended in the direction of the vehicle’s heading. It is assumed that the road direction is about in the same direction as the vehicle’s heading. Although there might be large gaps between two nearby curb cells, the extending process can fill the gap between the curb cells with negligible computational cost. This process significantly reduces the computational cost required in the connecting process.

For every cell $c_{i,j}$ (i row and j column), if it represents a curb and the vehicle is in the heading of θ , the $e_{q,w}$ are located and marked as extended curb cells for all $t = 1 \sim T$:

$$q = i + t * \cos(\theta)$$

$$w = j + t * \sin(\theta)$$

The constant T is the maximum extending distance, which is experimentally selected.

To connect the curb cells (after the extension), the curb map is scanned. For all the cells $c_{i,j}$ (i row and j column), if it represents a curb cell and the nearby cells are also curb cells, the cells between them are then marked as curb cells.

We choose a RANSAC algorithm to fit a line through the potential curb cells. The line fitting algorithms to the left (and similarly the right) curb cells proceeds as follows:

1. In the left frontal area of the vehicle, all the potential curb cells are organized into one list.
2. From this list, randomly select two cells $c1(a1,b1)$ and $c2(a2,b2)$, obtain the vector $c21(a2 - a1, b2 - b1)$ that describes a line through the points $(a1,b1)$ and $(a2,b2)$.
3. Calculate the distance to this line $c21$ from all the other cells $c(a,b)$ in the list:

$$d = \left| \frac{2 * ((a - a1) * (b2 - b1) - (b - b1) * (a2 - a1))}{|c21|} \right|$$

If d is smaller than a pre-selected threshold, the point $c(a,b)$ is considered to be on the line $c12$ and hence the line weight or the vote is increased by one.

Repeat the process from 2, to 3, for N ($=20$) randomly chosen pairs of points from the cell list. Now select the line with the highest weight (or voting) as the best fit curb line through the left curb cells.

The process is repeated with the curb cells to the right of the vehicle to find the best fit right curb line.

The vehicle is driven through Nanyang Technological University (NTU) campus roads and then curbs are detected and mapped. The approximated curb lines are shown (Figure 7-6) in white colour.

Figure 7-6: GUI view of curb line (white) extracted.

APPENDIX B: SAMPLE CUDA SIMD PROGRAM

B.1. Introduction

To illustrate how CUDA SIMD parallel computing kernel is developed, a simple example of matrix summation is presented [67]. Assume we have two matrix A and B of same dimensions $N \times N$, let's calculate

$$C = A + B$$

In CUDA, each program running in GPU is called a 'kernel', or a 'grid'. It is a small size function which can be called from the main() function. A kernel can be running in parallel by defining its number of blocks and number of threads in each block. The index of both blocks and threads can be 1,2,or 3 dimensional vector as demonstrated in figure 8-1.

Figure 8-1: CUDA architecture with block and thread.

B.2. Solution 1- Normal C/C++ program

The normal C++ program will calculate “C=A+B” by nested *for* loops:

```
//normal C++ function for C=A+B
float A[N] [N], B [N] [N], C[N] [N];
int main()
{
    for (int i=0;i<N;i++)
    {
        for (int j=0;j<N;j++)C[i][j]=A[i][j]+B[i][j];
    };
    return 0;
}
```

B.3. Solution 2 - CUDA with single block and multiple threads.

In CUDA, **threadIdx** is a 3-component vector, so that threads can be identified using a one-dimensional, two-dimensional, or three-dimensional *thread index*, forming a one-dimensional, two-dimensional, or three-dimensional *thread block*. This provides a natural way to invoke computation across the elements in a domain such as a vector, matrix, or field. As an example, the following code adds two matrices *A* and *B* of size *NxN* and stores the result into matrix *C*:

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 dimBlock(N, N);
    MatAdd<<<1, dimBlock>>>(A, B, C);
}
```

The index of a thread and its thread ID relate to each other in a straightforward way: For a one-dimensional block, they are the same; for a two-dimensional block of size

(Dx, Dy), the thread ID of a thread of index (x, y) is $(x + y \text{ Dx})$; for a three dimensional block of size (Dx, Dy, Dz), the thread ID of a thread of index (x, y, z) is $(x + y \text{ Dx} + z \text{ Dx Dy})$.

Threads within a block can cooperate among themselves by sharing data through some shared memory and synchronizing their execution to coordinate memory accesses. More precisely, one can specify synchronization points in the kernel by calling the `__syncthreads()` intrinsic function; `__syncthreads()` acts as a barrier at which all threads in the block must wait before any is allowed to proceed.

For efficient cooperation, the shared memory is expected to be a low-latency memory near each processor core, much like an L1 cache, `__syncthreads()` is expected to be lightweight, and all threads of a block are expected to reside on the same processor core. The number of threads per block is therefore restricted by the limited memory resources of a processor core. On current GPUs, a thread block may contain up to 512 threads.

B.4. Solution 3 - CUDA with multiple blocks and multiple threads.

A kernel can be executed by multiple equally-shaped thread blocks, so that the total number of threads is equal to the number of threads per block times the number of blocks. These multiple blocks are organized into a one-dimensional or two-dimensional grid of thread blocks as illustrated by Figure 2-1. The dimension of the grid is specified by the first parameter of the `<<<...>>>` syntax. Each block within the grid can be identified by a one-dimensional or two-dimensional index accessible within the kernel through the built-in `blockIdx` variable. The dimension of the thread block is accessible within the kernel through the built-in `blockDim` variable. The previous sample code becomes:

```

// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}
int main()
{
    ...
// Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x, (N + dimBlock.y - 1) /
dimBlock.y);
    MatAdd<<<dimGrid, dimBlock>>>(A, B, C);
}

```

The thread block size of $16 \times 16 = 256$ threads was chosen somewhat arbitrarily, and a grid is created with enough blocks to have one thread per matrix element as before. Thread blocks are required to execute independently: It must be possible to execute them in any order, in parallel or in series. This independence requirement allows thread blocks to be scheduled in any order across any number of cores, enabling programmers to write code that scales with the number of cores. The number of thread blocks in a grid is typically dictated by the size of the data being processed rather than by the number of processors in the system, which it can greatly exceed.