Using Reinforcement Learning in Multi-Robot SLAM

by

Pierre Dinnissen, **B.A.Sc.**

A thesis submitted to the

Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

September 14, 2011

©Copyright

Pierre Dinnissen, 2011

The undersigned hereby recommends to the

Faculty of Graduate and Postdoctoral Affairs

acceptance of the thesis

Using Reinforcement Learning in Multi-Robot SLAM

submitted by **Pierre Dinnissen**, **B.A.Sc.**

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical Engineering

Professor Howard Schwartz, Thesis Supervisor

Professor Sidney Givigi, Thesis Co-supervisor

Professor Howard Schwartz, Chair,
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

September, 2011

Abstract

Truly autonomous mobile robots require the ability to map their environment using 'Simultaneous Localization and Mapping' SLAM in order to explore it without getting lost. Multi-robot SLAM becomes necessary once an environment becomes too large.

This thesis focuses on the determination of when a robot should merge its maps with another robot's upon rendezvous. This decision should be based on the current status of the mapping filters and current status of the environment.

First, necessary software tools were required to be developed to support the research being done in this thesis. This includes the development of a simulated mobile robot dataset generator and modification of existing SLAM software.

Using Reinforcement Learning, the robot is trained to determine when to merge its maps and by which method to estimate the necessary transformation matrix. This allows the robot to incur less error than if it had simply merged upon first observing another robot.

In simulated experiments, we demonstrated that our approach allows multiple robots to map large environments that are difficult to map using a single robot.

Acknowledgments

I would like to acknowldege and thank my thesis advisor Prof. Howard Schwartz and Prof. Sidney Givigi for their support, guidance, and mentorship throughout my studies. I would like to give additional gratitude to all the professors that I have had the pleasure of studying with here at Carleton University. A special thanks to Xiaosong Lu for entertaining conversations, studying advice, and being a valuable sounding board.

I would also like to thank Prof. Joshua Marshall for offering the course for which I owe all of my knowledge of mobile robotics. I would also like to express my appreciation to everyone I have had the pleasure of meeting and interacting with here at Carleton, you truly make this institution proud. Finally, I would like to thank Lianne Lower for her unwavering support and patience during the difficult long hours and joyous breakthroughs.

Table of Contents

Abstract iii

Acknowledgments iv

Table of Contents v

List of Tables viii

List of Figures ix

List of Algorithms xii

Nomenclature xiii

1 Introduction 1

1.1 Problem Statement 2

1.2 Thesis Contributions 3

1.3 Organization of Thesis 4

2 Background and Literature Review 6

2.1 Particle Filters 6

2.2 Rao-Blackwellized Particle Filter Mapping 11

2.2.1 Different Map Types 12

2.2.2 FastSLAM 14
2.3 Reinforcement Learning 25
2.3.1 Learning in Mapping 26
2.4 Map Merging 28

3 Software Resources Development **34**
3.1 Mobile Robot Data Simulation Engine 34
3.1.1 Creation of Environments 35
3.1.2 Simulated Sensors 36
3.1.3 Trajectory Tracking 41
3.2 MRPT modifications 46
3.2.1 2D Feature Map Type 47
3.2.2 Map Merging 47
3.2.3 Other Robot Observation Type 48
3.2.4 DataSet to Rawlog Converter 49
3.2.5 Particle Filter Modifications 50

4 Proposed Approach **52**
4.1 Calculation of Transformation Matrix 52
4.1.1 Improving Transformation Matrix Estimate 56
4.2 Model Selection 60
4.2.1 N_{eff} Criterion 61
4.2.2 Mutual Observation Likelihood Criterion 62
4.2.3 Reinforcement Learning for Model Selection 64

5 Simulations **73**
5.1 Results 74

5.2 Validation 77
5.3 Single-Robot vs Multi-Robot SLAM 82

6 Conclusions **86**
6.1 Summary of Contributions 87
6.2 Future Work 89

List of References **91**

Appendix A Derivation of Differential Drive Robot Controller **94**

Appendix B MRPT Code Modifications **99**
B.1 C2DFeature Header File 100
B.2 C2DFeatureMap Header File 101
B.3 C2DFeatureMap Map Merging Code 102
B.4 COccupancyGridMap2D Map Merging Code 104
B.5 CObservationOtherRobot Header File 105

Appendix C MRPT Particle Filter Parameters **106**
C.1 Config File 107

List of Tables

3.1 Simulated Laser Range Finder Parameters 39
3.2 Simulated Feature Detector Parameters 41
3.3 Simulated Other Robot Parameters 42
3.4 Sensor to MRPT class mapping 50

List of Figures

2.1 Illustration of the main steps of a particle filter. 7

2.2 Example of propagation of PDF and $N = 10$ particles 9

2.3 Example of propagation of PDF and $N = 100$ particles 10

2.4 Examples of map types used in this thesis. 14

2.5 Derivation of feature measurement model 18

2.6 Illustration on how the Likelihood Field Range Finder Model is derived 21

2.7 Example of a likelihood field lookup table. 21

2.8 Flowchart showing steps to merge feature maps 32

2.9 Flowchart showing steps to merge occupancy grid maps 33

3.1 Simple Indoor and Outdoor Environment 35

3.2 Sample LRF scan measurement 40

3.3 Sample feature detector measurement 40

3.4 Sample other robot measurement 42

3.5 Diagram showing the state variables of a differential drive vehicle . . 43

3.6 Controller Architecture for Trajectory Tracking 45

3.7 Trajectory Tracking Result 46

4.1 Diagram illustrating how the relative equations are determined. . . . 55

4.2 Diagram illustrating how transformation parameters and matrices have been determined. 55

4.3 Diagram illustrating how the LSE can be used to improve the transformation matrix. 57
4.4 Diagram illustrating how the ICP can be used to improve the transformation matrix. 59
4.5 Diagram demonstrating the process of how the maps produced from FastSLAM by each individual robot are merged from one robot's perspective. 61
4.6 State Diagram demonstrating how the Reinforcement Learning states transition between one another. 66
4.7 Training Environment with building and set of trees with dimensions 45 [m] ×45 [m] 70
5.1 Decision Tree showing the policy determined through reinforcement learning 75
5.2 Comparison of cumulative error for one entire training data robot trajectory 76
5.3 Comparison of cumulative error for another entire training data robot trajectory 77
5.4 Environment used for validation with dimensions 45 [m] ×55 [m] . . . 78
5.5 Comparison of cumulative error for one entire validation data robot trajectory 79
5.6 Comparison of cumulative error for one entire validation data robot trajectory 80
5.7 Sample maps produced during validation 81
5.8 Sample maps produced during validation 81
5.9 Trajectories used in comparing a single robot mapping compared to multi-robot mapping 83

5.10 Comparison of cumulative error from Single Robot and Multi-Robot SLAM 84
5.11 Sample maps produced during a single robot run 84
5.12 Sample maps produced by two robots that merged midway 85

List of Algorithms

2.1 Basic Particle Filter [5] 7

2.2 Simplest Resampling Method [5] 11

2.3 Generalized Basic steps of FastSLAM [5] 15

2.4 Improved Feature Mapping Pose Sampling [9] 16

2.5 Improved Grid Mapping Pose Sampling [10] 17

2.6 Likelihood Field Range Finder Model [5] 20

2.7 Integration of feature measurements in feature maps [5] 23

2.8 Integration of LRF data in occupancy grid maps [5] 24

2.9 SARSA Algorithm [12] 26

2.10 Wurm's Combined Mapping Approarch [1] 29

4.1 Mapping and Map-Merging approach 71

4.1 *Continued* 72

Nomenclature

Abbreviations:

SLAM	Simultaneous Localization and Mapping
EKF	Extended Kalman Filters
RBPF	Rao-Blackwellized Particle Filter
PDF	probability density function
LRF	Laser Range Finder
TD	Temporal-Difference
MRPT	Mobile Robot Programming Toolkit
ICP	Iterative Closest Point
LSE	Least Squares Estimate

Mobile Robots:

t	time in seconds
S_t	Sample set of particles at time t
x_t	pose of a robot $(x, y, \theta)^T \in \mathbb{R}^2 \times \mathbb{S}^1$ at time t
$x_t^{(i)}$	estimate of the pose of a robot in particle i
N	number of particles used in the filter
u_t	measured control signal, i.e. system input, at time t
z_t	sensor measurement observed at time t
$p(a\|b)$	probability of a given b
$w_t^{(i)}$	weight or relative likelihood of a particle i at time t

$\mathcal{N}(\mu, \Sigma)$	Gaussian/Normal distribution function with mean μ and covariance Σ
R	measurement noise model covariance
η	normalizing coefficient for Gaussian distribution
h	measurement model for sensor
m	map representing the environment
$y_{1:t}$	combined state vector $\begin{pmatrix} x_{1:t} \\ m_{1:t} \end{pmatrix}$ from initialization to timestep t
m_g	occupancy grid map
m_f	feature map
λ^j	log odd value of cell at index j
c_j	single cell at index j of an occupancy grid map
z_g	measurement from a laser range finder
$\mu_{j,t}^{(i)}$	mean coordinate location (x, y) of feature j at time t in particle i
$\Sigma_{j,t}^{(i)}$	covariance of coordinate location (x, y) of feature j at time t in particle i
Q	control signal measurement noise covariance matrix
g	vehicle model for mobile robot
r	distance representing range from sensor to a feature or reward signal in context with re
ϕ	angle representing bearing from sensor x-axis to a feature
H_m	Jacobian of h w.r.t. feature coordinate location
H_{x_t}	Jacobian of h w.r.t. robot pose
N_{eff}	effective sample size value
x_t^*	true robot pose at time t
T	transformation matrix
q	pose of a robot $(x, y, \theta)^T \in \mathbb{R}^2 \times \mathbb{S}^1$
v_R	forward velocity of the right wheel
v_L	forward velocity of the left wheel
v	forward velocity of the robot
ω	angular veloctity of the robot
l	distance between robot wheels

Reinforcement Learning:

S	reinforcement learning state
A	available action set
r	reward signal for policy update
$Q(s, a)$	action-value function
α	learning rate
γ	discounting rate
$\bar{l}$	average scan matching likelihood for [1]
a_g	choose grid-based method for proposal sampling
a_f	choose feature-based method for proposal sampling
N_{eff}^c	effective sample size for the current robot's particle filter
N_{eff}^o	effective sample size for the other robot's particle filter
$\bar{l}_g$	confidence of the ICP recommended transformation matrix
$\bar{l}_f$	confidence in the LSE recommended transformation matrix
$a_m f$	merge using feature-based transformation matrix estimate
$a_m g$	merge using grid-based transformation matrix estimate
$a_m i$	merge using initial transformation matrix estimate

Chapter 1

Introduction

Autonomous vehicles have a great many applications ranging from underground mining to planetary exploration. To accomplish complete autonomy in vehicles, many problems must be solved. These problems are often categorized into guidance, navigation, and control. Guidance involves intelligent decision making of where the vehicle should go and which path or trajectory should be taken to reach the goal. Navigation is the ability of the robot to perceive or localize where it currently is in the environment in order to ensure it is on the proper course. Finally, control is the method by which the robot determines the signals to send to the vehicle actuators.

Guidance and navigation systems require there to be a spatial model of the environment in order to allow true autonomous robots to independently explore and traverse the environment. Despite years of research in this field, there remains many issues left to be resolved such as mapping unstructured, dynamic, or large-scale environments. The mobile robot mapping problem requires a circular reference solution where the localization of the robot must be known to build an accurate map and an accurate map is required to best estimate the localization of the robot. This is best known as *Simultaneous Localization and Mapping (SLAM)*. Initial solutions focussed on the use of Extended Kalman Filters (EKFs) [2], but since this time many different

solutions have been developed. These solutions range from sparse extended information filters to particle filters. However, only particle filter based SLAM algorithms will be used in this thesis.

Many of these techniques focus on a specific representation of the environment. In a sparse outdoor environment, a feature-based model using landmark coordinates is most beneficial. Conversely, in a highly structured indoor environment it is best to use a dense representation such as an occupancy grid map. In practice, robots should be capable of mapping any type of environment that could potentially contain both indoor and outdoor components. Thus, solutions capable of handling these more complex situations need to be developed. Wurm, Stachniss, and Grisetti [1] have applied reinforcement learning to determine which map representation should be used to determine the localization of the robot depending on the current sensor observations. The major contribution of this thesis is to extend this novel approach to multi-robot mapping applications.

Computational efficiency and map accuracy begin to suffer once an environment becomes too large. A logical solution is to use multiple robots in what is known as Multi-Robot SLAM. Burgard *et al.* first developed a technique of merging maps under the constraint that the initial relative poses between robots were known [3]. In the Zhou and Roumeliotis scenario, two robots meet each other without prior knowledge of their respective starting poses [4].

1.1 Problem Statement

The task of this thesis is not to develop a more improved map-merging algorithm but to develop a decision algorithm capable of deciding when to merge and which map representation to use in attempting to reduce the incurred error from map merging.

In order to reach the best policy, sufficient data is required and can be produced through computer simulation. The computer simulates an environment composed of a building made of walls and an outdoor environment composed of trees. The walls are represented as line segments and the outdoor environment is represented by trees which are modeled as circles. Datasets produced from this simulated environment are then processed by both the training algorithm and the validation testing.

To successfully map this indoor and outdoor environment, the work in dual representation SLAM by Wurm *et al.* [1] will need to be reproduced and validated. The training algorithm determines when and how to best merge the maps of the individual robots. The algorithm being produced is a decentralized decision making process, which means the decision to merge one robot's map with another is performed by the current robot. A centralized decision making process would involve some type of higher 'authority' commanding when the robots should merge their maps. To avoid confusion, for the rest of this thesis the algorithm is described from the viewpoint of one of the robots.

1.2 Thesis Contributions

The main contribution of this thesis is to determine the best policy for map merging using reinforcement learning that reduces the amount of error incurred through map merging. Reinforcement Learning is a form of machine learning where actions are taken in an environment in order to maximize the given reward. To accomplish this goal other smaller contributions had to be performed first. These other contributions are the development of a simulation engine, adaptation of existing SLAM software, and the reproduction and validation of the work of Wurm *et al.* [1].

Availabe dataset simulation engines did not meet the accuracy and flexibility

requirements of this thesis. Therefore, a suitable simulation engine was developed in MATLAB that was capable of accepting a list of building lines and feature circles along with a desired trajectory path for a simulated robot and produce realistic mobile robot datasets. The ease of programming and debugging abilities available in MATLAB allowed this simulation engine to be developed and used quickly.

However, MATLAB's computational speeds became insufficient when it came to processing these simulated datasets. Thus, a C++ open-source library known as MRPT (Mobile Robot Programming Toolkit) was extended and used as necessary. The extensions included adding a basic type of two dimensional feature map, adding map merging functions, and parallelizing some functions to take advantage of multi-core CPUs.

Another minor yet necessary contribution of this thesis was the validation of the work done in [1] where reinforcement learning was used to determine which map representation should be used in selecting the appropriate proposal distribution to sample new particles.

1.3 Organization of Thesis

In the next chapter, background topics necessary to produce maps from the datasets is described. Particle filters are explored and shown how they can be adapted to be used to solve the SLAM problem. This is followed by a literature review of the advanced methods using these adapted particle filters named Rao-Blackwellized Particle Filters (RBPF). Next, the basics of reinforcement learning is discussed, followed by a description of how it can be used in robot mapping. Finally, a brief literature review is presented showing the work that has been done in map merging.

Chapter 3 covers the development of the simulation engine used to produce realistic mobile robot datasets. This includes a description of the simulated sensors and control methods used to accomplish realistic mobile robot motion. Additionally, the work done in expanding and adapting the MRPT library is detailed.

In chapter 4, the proposed approach is presented. We show how reinforcement learning can improve map merging efficiency and accuracy. The methods used to determine the transformation matrices for map merging and how to improve them are outlined. The model used in the reinforcement learning training is also detailed. How datasets are created to ensure that each state in the reinforcement learning model are visited sufficiently is demonstrated.

The work done in establishing a proper training environment is described in chapter 5. Validation of the results of the training is also demonstrated.

Finally, chapter 6 discusses the conclusions regarding the content of this thesis and outlines possible directions of future work.

Chapter 2

Background and Literature Review

2.1 Particle Filters

A particle filter is a 'brute force' numerical implementation that is capable of solving the Bayesian estimator problem nonparametrically. This method only became feasible due to advances in cheap and powerful computers. Although this implies that particle filter algorithms are computationally inefficient, variations can be designed that mitigate this defiency. The fundamental steps of a particle filter algorithm are shown in Algorithm 2.1 and illustrated in Figure 2.1. In Figure 2.1, particles are shown as black dots. In the weight calculation step shown in Figure 2.1, higher weighted particles are shown larger and lower weighted particles are shown smaller.

Particles are samples of the posterior distribution and are denoted as

$$S_t := x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(N)} \tag{2.1}$$

where $x_t^{(N)}$ is a particle representing the hypothesis of the true world state at time t and N is the number of particles. For best results, the number of particles should be

Algorithm 2.1 Basic Particle Filter [5]

Require:
S_{t-1}, previous time step particle sample set
u_t, most recent control
z_t, most recent measurement
Ensure:
S_t, new sample set

$S_t = \bar{S}_t = \{\}$ // *Start with Empty Set*
for $i = 1 \rightarrow N$ **do**
  $x_t^{(i)} \sim p(x_t | x_{t-1}^{(i)}, u_t)$ // *Sample New Particle*
  $w_t^{(i)} = p(z_t | x_t^{(i)})$ // *Calculate Weight of Particle*

  $\bar{S}_t = \bar{S}_t \cup \{\langle x_t^{(i)}, w_t^{(i)} \rangle\}$ // *Update Sample Set*
end for

for $i = 1 \rightarrow N$ **do** // *Resample*
  draw j with probability $\propto w_t^{(j)}$
  $S_t = S_t \cup \langle x_t^{(j)} \rangle$
end for

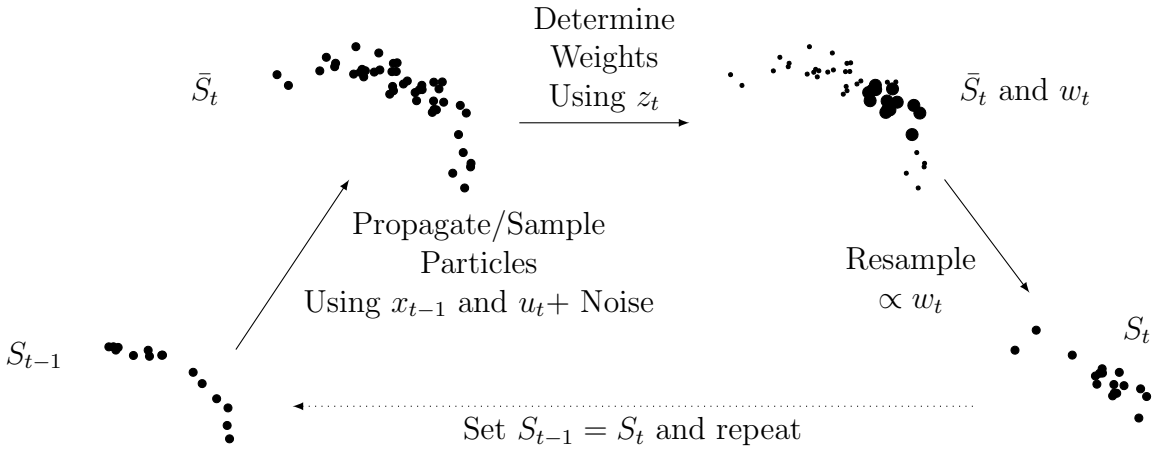


Figure 2.1: Illustration of the main steps of a particle filter.

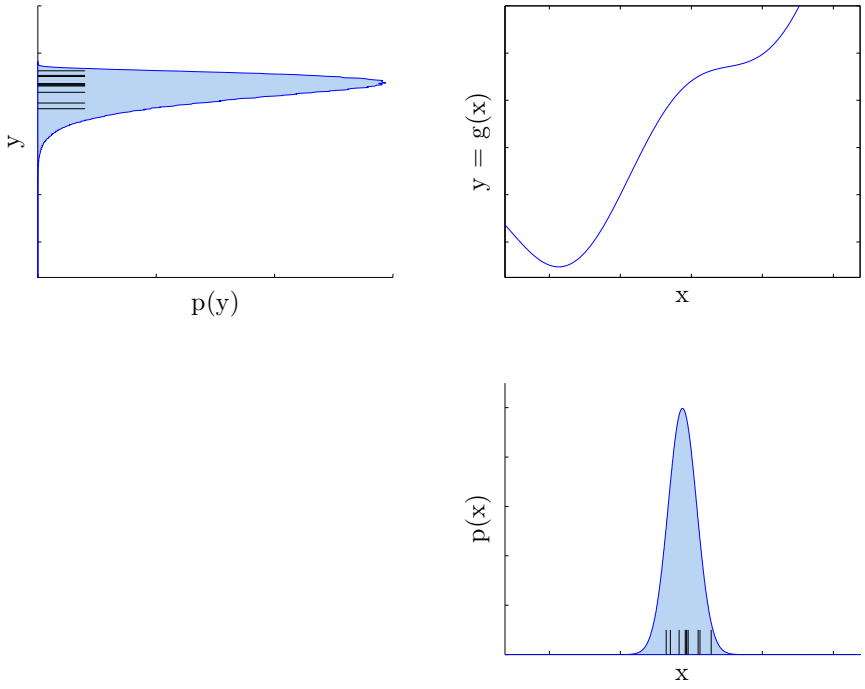
large, e.g. $N > 1000$, or it can be based on a function of t or vary upon the belief of the current state. There is an obvious trade off between the number of particles and the accuracy of the filter. In the case of mobile robots, the number of particles should be based on the variance of the measurable actuator signals or odometry estimates. If the variance is small then a smaller number of particles should be required to accurately represent the posterior. Consequently, if the variance is large then a larger number of particles should be chosen, as is illustrated in Figure 2.2 and Figure 2.3.

The bottom right plots of the subfigures in Figure 2.2 and Figure 2.3 show a probability density function (PDF), $p(x)$, and individual particles. The top right plots show the nonlinear function $y = g(x)$ that the PDF and particles are propagated through. Finally, the top left plots show the propagated PDF, $p(y)$, and propagated particles.

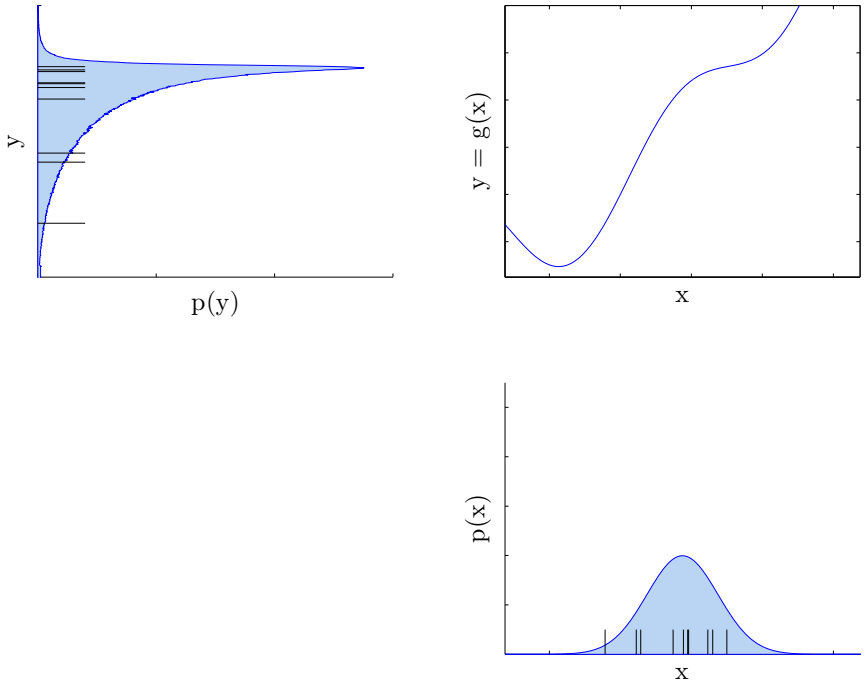
The particle sample set S_{t_0} should be initialized according to the particular belief of the initial state x_{t_0}. For example, trying to localize a robot within a map without any knowledge of initial starting position. In this case, it would be appropriate to randomly select a particle set that attempts to cover as many possible poses within the map area as possible. Conversely, if the determination of the state is relative to the starting position of the robot, it would be best to simply initialize all particles to the origin.

To propagate or sample new particles, there needs to be a vehicle model, $x_t = f(x_{t-1}, u_t, n_t)$, and an assumption of the probability density function of the control error. Although it is a common assumption for the error in observed signals to be Gaussian, this noise model can be of any shape.

The calculation of the weights, w_t, or *relative likelihood*, of each particle is accomplished using some type of measurement, z_t, and its probability density function

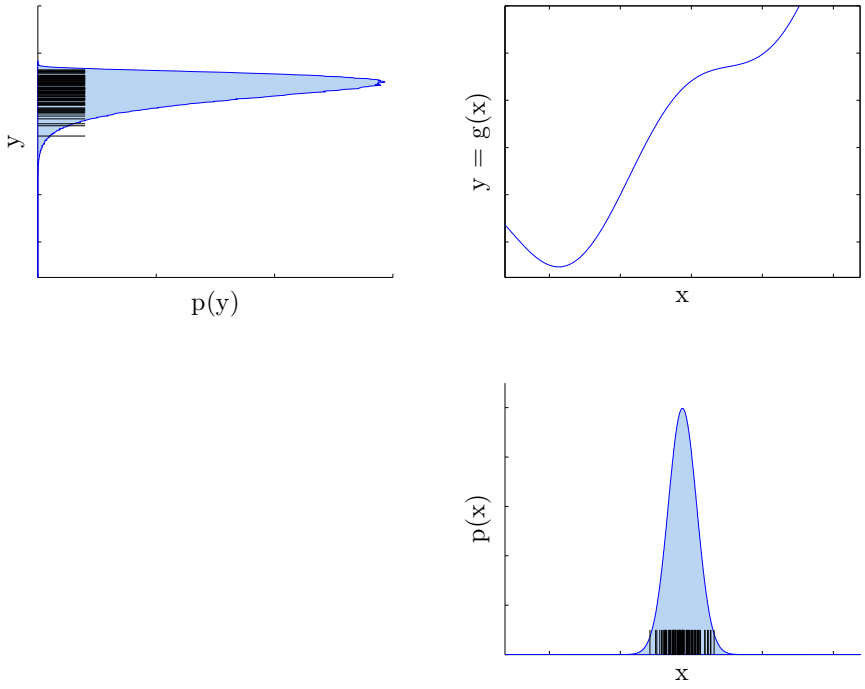


(a) Small Variance

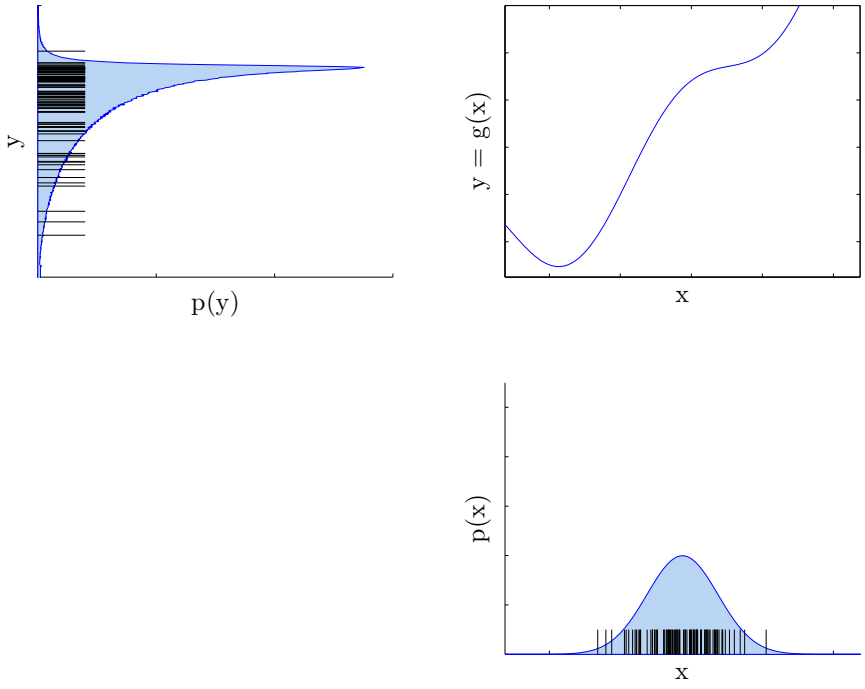


(b) Large Variance

Figure 2.2: Example of propagation of PDF and $N = 10$ particles



(a) Small Variance



(b) Large Variance

Figure 2.3: Example of propagation of PDF and $N = 100$ particles

(PDF). Typically and for this thesis, the measurement noise is deemed to be Gaussian, $\mathcal{N}(0, R)$. For which, the weights are calculated as follows

$$w_t^{(i)} = \eta \exp\left(-\frac{1}{2}(z_t - h(x_t^{(i)}))R^{-1}(z_t - h(x_t^{(i)}))\right) \tag{2.2}$$

where η is the normalizing coefficient, $h(x_t^{(i)})$ is the expected measurement based on the current particle state estimate. All weights should be normalized as follows

$$w_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^{N} w_t^{(j)}} \tag{2.3}$$

Finally the correction step of the particle filter is the resampling step, which can also be accomplished in several different ways. One of the simplest methods is described in Algorithm 2.2.

Algorithm 2.2 Simplest Resampling Method [5]

for $i = 1 \rightarrow N$ **do**
Draw random number ρ from uniform distribution $[0, 1]$
Accumulate sum of weights until $\sum_{m=1}^{i-j} w_t^{(m)} < \rho$ but $\sum_{m=1}^{j} w_t^{(m)} \geq \rho$, save j
$S_t = S_t \cup \langle x_t^{(j)} \rangle$
end for

One of the strengths of particle filters is that they can be extended to provide a solution to a specific problem, as will be shown in the next section.

2.2 Rao-Blackwellized Particle Filter Mapping

The Rao-Blackwellized Particle Filter (RBPF) is currently one of the most popular mapping techniques where particles are used to represent the posterior probability, which is the conditional probability assigned once relevant evidence has been taken

into account, over some variables and some parametric PDF to represent all other variables [6]. Montemerlo *et al.* used this technique in the development of a simple yet effective feature-based mapping method known as FastSLAM [7]. In FastSLAM, each particle contains a possible trajectory of the robot and a map of the environment. It can be mathematically shown [5] that for each particle, individual map error is conditionally independent and thus the mapping problem can be delegated to many separate problems. In terms of the full SLAM posterior estimate, this factorization can be written mathematically as:

$$p\left(y_{1:t}|z_{1:t},u_{1:t}\right)=p\left(x_{1:t}|z_{1:t},u_{1:t}\right)\prod_{n=1}^{N}p\left(m_n|x_{1:t},z_{1:t}\right) \tag{2.4}$$

where $y_{1:t}$ is the combined state vector $\begin{pmatrix} x_{1:t} \\ m \end{pmatrix}$, $x_{1:t}$ is the pose of the robot, m is the map, $z_{1:t}$ are the environment measurements, and $u_{1:t}$ are the robot controls.

2.2.1 Different Map Types

Different types of maps m must be chosen depending on the environment being explored. In this thesis, occupancy grid maps m_g and feature maps m_f will be used.

Occupancy grid maps are composed of an array of cells containing the likelihood of a cell being occupied and are best suited for structured environments such as buildings. The likelihoods of individual cells are actually stored as *log odds* which have the advantage over probability representation of avoiding instabilities for probabilities near zero or one. A log odd for each cell indexed j can be computed by

$$\lambda_t^j=\log\frac{p(c_{j,t}|z_{g,1:t},x_{1:t})}{1-p(c_{j,t}|z_{g,1:t},x_{1:t})} \tag{2.5}$$

where λ_t^j is the log odd, $c_{j,t}$ is an individual occupancy grid cell, $z_{g,1:t}$ is all the laser range finder (LRF) measurements, and $x_{1:t}$ is the complete trajectory of the robot. To recover the probabilites from the log odd, the following equation can be used

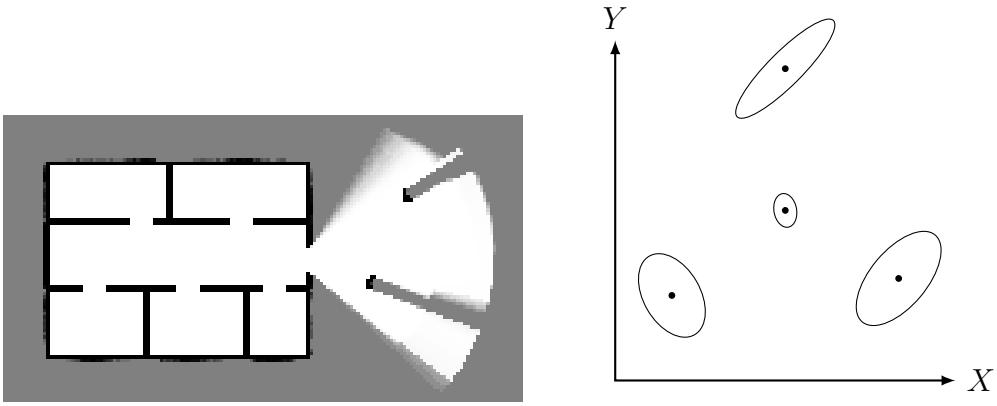
$$p(c_{j,t}|z_{g,1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp(\lambda_t^j)} \tag{2.6}$$

When evidence is found that a cell is either *free* or *occupied*, then a predetermined constant, corresponding to the type of evidence found, is added to the current log odd value of that cell to update it accordingly.

Feature maps are composed of a vector of mean coordinate locations, $\mu_{\#,t}^{(i)}$, and covariances, $\Sigma_{\#,t}^{(i)}$, representing each individual feature and are best used in a sparse outdoor environment containing multiple trees and poles. The locations and covariances are updated using a simple EKF for each feature.

Figure 2.4(a) shows an example of an occupancy grid map that has been converted to a grayscale image. Typically, each grayscale image pixel is 8-bits, which means it can store values ranging from $0 \rightarrow 255$. To convert an occupancy grid map to an image, the log odds must first be converted to probabilities ranging from $0 \rightarrow 1$ which can then be mapped to the available range of pixel values. Visually, it makes more sense to use the pixel value of 0, or black, to represent an occupied cell which has a corresponding probability of 1. Conversely, a pixel value of 255, or white, will represent a free cell well. Cells that have received insufficient or no evidence will be represented by some shade of gray.

Figure 2.4(b) shows a simple feature map. The estimated position of a feature is shown by a dot and the corresponding covariance of uncertainty is shown as an ellipse.



(a) Occupancy Grid (b) Feature Map

Figure 2.4: Examples of map types used in this thesis.

2.2.2 FastSLAM

As previously mentioned, FastSLAM uses a particle filter to solve the SLAM problem. This is accomplished by having each individual map feature location estimated by an EKF. The resulting algorithm can be implemented in logarithmic time in terms of the number of features; allowing this algorithm to solve the full SLAM problem in real-time, also known as the *online SLAM problem* [5].

One of the key advantages of FastSLAM is its extensibility. Although originally designed for feature based environments, Hahnel *et al.* adapted FastSLAM for grid-based maps that could be produced with only the raw laser range finder data [8].

FastSLAM requires a great number of particles due to the fact that the accuracy of the control observation is typically low relative to the accuracy of the measurement sensors. In other words, the proposal distribution used in sampling for new particles is based only on the previous pose x_{t-1} and the control signal u_t. FastSLAM 2.0 [9] improves this proposal distribution by additionally incorporating z_t in the proposal distribution. This modification leads to a more efficient algorithm requiring fewer particles to achieve the same results, but also leads to a more complex implementation

[9]. Grisetti *et al.* used this idea of improved proposal distribution in grid-based FastSLAM [10]. The basic steps of a generic FastSLAM algorithm are shown in Algorithm 2.3. The key difference between Algorithm 2.3 and Figure 2.1 is the use of a resampling condition which is discussed later in the text.

Algorithm 2.3 Generalized Basic steps of FastSLAM [5]

Require:
$S_{t-1} = \{\langle x_{t-1}^{(1:n)}, w_{t-1}^{(1:n)}, m_{t-1}^{(1:n)}\rangle\}$, previous time step sample set
z_t, most recent sensor observation
u_{t-1}, most recent odometry measurement
Ensure:
S_t, new sample set

$S_t = \{\}$ // *Start with Empty Set*
for all $s_{t-1}^{(i)} \in S_{t-1}$ **do**
  $\langle x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)}\rangle = s_{t-1}^{(i)}$ // *Retrieval*
  $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1}, m_{t-1}^{(i)}, z_t)$ // *Sample new Particle*
  $w_t^{(i)} = \text{updateWeight}(w_{t-1}^{(i)}, m_{t-1}^{(i)}, z_t^{(i)})$
  $m_t^{(i)} = \text{integrateSensorObs}(m_{t-1}^{(i)}, x_t^{(i)}, z_t^{(i)})$

  $S_t = S_t \cup \{\langle x_t^{(i)}, w_t^{(i)}, m_t^{(i)}\rangle\}$ // *Update Sample Set*
end for

if Resampling Condition Met **then**
  $S_t = \text{resample}(S_t, \{w^{(i)}\})$
end if

Improving Sample Proposal Distribution

The improved versions of FastSLAM depend on reducing the covariance of the sample proposal distribution for new particles, $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1}, z_t)$. Algorithm 2.4 from [9] and Algorithm 2.5 from [10] respectively show how the measurements $z_{f,t}$ and $z_{g,t}$ are included to produce a smaller sampling distribution for each mapping algorithm.

Algorithm 2.4 Improved Feature Mapping Pose Sampling [9]

Require:
$x_{t-1}^{(i)}$, pose from previous time step
u_{t-1}, most recent odometry measurement
$m_{f,t-1}^{(i)} = \left\langle \langle \mu_{1,t-1}^{(i)}, \Sigma_{1,t-1}^{(i)} \rangle, \cdots, \langle \mu_{F,t-1}^{(i)}, \Sigma_{F,t-1}^{(i)} \rangle \right\rangle$, feature map from previous time step with F number of features
$z_{f,t}$, most recent feature measurement
Q, measurement noise covariance matrix
g, vehicle model
h, measurement model
Ensure:
$x_t^{(i)}$, sampled proposed pose

$\hat{x}_t = g(x_{t-1}^{(i)}, u_{t-1})$ // *predict pose based on vehicle model*
$\mu_{x_t} = \hat{x}_t$
$\Sigma_{x_t} = P_{x,t}$ // *pose covariance*
for all features j that are both in $m_{t-1}^{(i)}$ and $z_{f,t}$ **do**
  $\bar{z}_j = h(\mu_{j,t-1}^{(i)}, \hat{x}_t)$ // *measurement prediction*
  $H_{x_t,j} = \nabla_{x_t} h(\mu_{j,t-1}^{(i)}, \hat{x}_t)$ // *Jacobian wrt pose*
  $H_{m,j} = \nabla_{m_j} h(\mu_{j,t-1}^{(i)}, \hat{x}_t)$ // *Jacobian wrt feature*
  $Q_j = Q + H_{m,j} \Sigma_{j,t-1}^{(i)} H_{m,j}^T$ // *measurement information*
  $\Sigma_{x_t} = \left[H_{x,j}^T Q_j^{-1} H_{x,j} + \Sigma_{x_t}^{-1} \right]^{-1}$ // *adjust Cov of proposal*
  $\mu_{x_t} = \mu_{x_t} + \Sigma_{x_t} H_{x,j}^T Q_j^{-1} (z_{f,t}^{(j)} - \bar{z}_j)$ // *adjust mean of proposal*
end for
$x_t^{(i)} \sim \mathcal{N}(\mu_{x_t}, \Sigma_{x_t})$ // *sample pose*

Algorithm 2.5 Improved Grid Mapping Pose Sampling [10]

Require:
$x_{t-1}^{(i)}$, pose from previous time step
u_{t-1}, most recent odometry measurement
$m_{g,t-1}^{(i)}$, grid map from previous time step
$z_{g,t}$, most recent LRF measurement
Ensure:
$x_t^{(i)}$, sampled proposed pose

$\hat{x}_t^{(i)} = \text{argmax}_x \, p(x|m_{g,t-1}^{(i)}, z_{g,t}, g(x_{t-1}^{(i)}, u_{t-1})$ // *scan-matching*
if $\hat{x}_t^{(i)} =$ failure **then**
  $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$ // *sample based on odometry measurement only*
else
  // *sample around an interval from the scan matcher pose*
  for $k = 1, \cdots, K$ **do**
    $x_k \sim \{x_j| \, |x_j - \hat{x}^{(i)}| < \Delta\}$ // *sample based on uniform distribution*
  end for

  // *compute Gaussian proposal*
  $\mu_{x_t} = (0, 0, 0)^T$
  $\eta_{x_t} = 0$
  for all $x_j \in \{x_1, \cdots, x_K\}$ **do**
    $\mu_{x_t} = \mu_{x_t} + x_j \cdot p(z_{g,t}|m_{g,t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    $\eta_{x_t} = \eta_{x_t} + p(z_{g,t}|m_{g,t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
  end for
  $\mu_{x_t} = \mu_{x_t}/\eta_{x_t}$
  $\Sigma_{x_t} = \mathbf{0}$
  for all $x_j \in \{x_1, \cdots, x_K\}$ **do**
    $\Sigma_{x_t} = \Sigma_{x_t}(x_j - \mu_{x,t})(x_j - \mu_{x,t})^T \cdot p(z_{g,t}|m_{g,t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
  end for
  $\Sigma_{x_t} = \Sigma_{x_t}/\eta_{x_t}$

  $x_t^{(i)} \sim \mathcal{N}(\mu_{x_t}, \Sigma_{x_t})$ // *sample pose*
end if

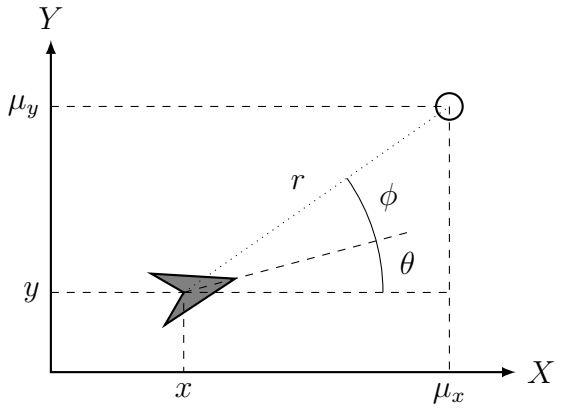


Figure 2.5: Derivation of feature measurement model

In Algorithm 2.4, the measurement prediction $\bar{z}_j = h(\mu_{j,t-1}^{(i)}, \hat{x}_t)$ depends on the measurement model. For this thesis, the measurement model h for feature observations is defined as:

$$h(\mu, x_t) = \begin{bmatrix} r \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(\mu_x - x)^2 + (\mu_y - y)^2} \\ \text{atan2}\,(\mu_y - y, \mu_x - x) - \theta \end{bmatrix} \tag{2.7}$$

where $(x, y, \theta)^T = x_t$ is the robot pose. The derivation of this model is shown in Figure 2.5, where a feature in the environment is represented by a circle.

The Jacobian of (2.7) with respect to feature position is

$$\begin{aligned} H_m = \nabla_m h(\mu, x_t) &= \begin{bmatrix} \frac{dr}{d\mu_x} & \frac{dr}{d\mu_y} \\ \frac{d\phi}{d\mu_x} & \frac{d\phi}{d\mu_y} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\mu_x - x}{\sqrt{(\mu_x - x)^2 + (\mu_y - y)^2}} & \frac{\mu_y - y}{\sqrt{(\mu_x - x)^2 + (\mu_y - y)^2}} \\ \frac{-(\mu_y - y)}{(\mu_x - x)^2 + (\mu_y - y)^2} & \frac{\mu_x - x}{(\mu_x - x)^2 + (\mu_y - y)^2} \end{bmatrix} \end{aligned} \tag{2.8}$$

and the Jacobian of (2.7) with respect to pose is

$$H_{x_t} = \nabla_{x_t} h(\mu, x_t) = \begin{bmatrix} \frac{dr}{dx} & \frac{dr}{dy} & \frac{dr}{d\theta} \\ \\ \frac{d\phi}{dx} & \frac{d\phi}{dy} & \frac{d\phi}{d\theta} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{-(\mu_x - x)}{\sqrt{(\mu_x - x)^2 + (\mu_y - y)^2}} & \frac{-(\mu_y - y)}{\sqrt{(\mu_x - x)^2 + (\mu_y - y)^2}} & 0 \\ \\ \frac{\mu_y - y}{(\mu_x - x)^2 + (\mu_y - y)^2} & \frac{-(\mu_x - x)}{(\mu_x - x)^2 + (\mu_y - y)^2} & -1 \end{bmatrix} \tag{2.9}$$

Weight Calculation for Particles

The function 'updateWeight' can vary depending on the implementation, but generally the weight of the particle is calculated by finding the likelihood of the most recent measurement at the sampled pose in the current status of the map [5]. In feature maps, this likelihood can be calculated using an equation similar to (2.2) for each feature currently observed and then averaged as follows

$$w_t^{(i)} = \frac{\sum_{j=1}^{F_{\text{mut}}} \exp\left(-\frac{1}{2}(z_{f,t}^{(j)} - \hat{z}_{f,t}^{(i,j)}) Q_t^{-1} (z_{f,t}^{(j)} - \hat{z}_{f,t}^{(i,j)})\right)}{F_{\text{mut}}} \tag{2.10}$$

where F_{mut} is the number of features that are both in the current measurement and the current map $m_{f,t}$; $z_{f,t}^{(j)}$ is the measurement of a current feature; $\hat{z}_{f,t}^{(i,j)} = h(\mu_{j,t-1}^{(i)}, x_t^{(i)})$ is the measurement prediction; and $Q_t = H\Sigma_{j,t-1}^{(i)} H^T + Q$ is the predicted measurement covariance with $H = h'(\mu_{j,t-1}^{(i)}, x_t^{(i)})$ being the Jacobian of the measurement model and Q is the measurement sensor noise covariance. Also, $\mu_{j,t-1}^{(i)}$ and $\Sigma_{j,t-1}^{(i)}$ are the mean and covariance of a feature location respectively and their methods of calculation is shown in Algorithm 2.7.

The measurement model h is given in (2.7) and the corresponding Jacobian H is

given by (2.8).

Weights in grid-based maps are calculated using the 'beam endpoint model' [5]. This method is an algorithm that generally works well, but does not actually compute a conditional probability relative to the actual physical model of the sensors. Algorithm 2.6 outlines how this likelihood is calculated and Figure 2.6 illustrates how this algorithm is derived.

Algorithm 2.6 Likelihood Field Range Finder Model [5]

Require:

$z_{g,t}$, most recent LRF data measurement
$x_t = (x, y, \theta)^T$, proposed pose
$m_{g,t}$, current status of occupancy grid map
ζ_{hit}, mixing weight for probability of ray hitting an obstacle
ζ_{rand}, mixing weight for probability of ray having an incorrect value
$z_{g,max}$, the constant maximum value of LRF
σ_{hit}, variance in the accuracy of LRF

Ensure:

τ, likelihood value

$\tau = 1$
for $j = 1 \rightarrow G$ **do** // *G is the total number of rays*
  if $z^j_{g,t} \neq z_{g,max}$ **then**
    $x_{z^j_{g,t}} = x + x_{j,\text{sens}} \cos\theta - y_{j,\text{sens}} \sin\theta + z^j_{g,t} \cos(\theta + \theta_{j,\text{sens}})$
    $y_{z^j_{g,t}} = y + y_{j,\text{sens}} \cos\theta - x_{j,\text{sens}} \sin\theta + z^j_{g,t} \sin(\theta + \theta_{j,\text{sens}})$
    $d = \min_{x',y'} \left(\sqrt{(x_{z^j_{g,t}} - x')^2 + (y_{z^j_{g,t}} - y')^2} \middle| \langle x', y' \rangle \text{ occupied in } m_{g,t} \right)$
    $\tau = \tau \cdot \left(\zeta_{hit} \cdot \exp\left(-\frac{d^2}{2\sigma^2_{hit}} \right) + \frac{\zeta_{random}}{z_{g,max}} \right)$
  end if
end for

The most computationally expensive part of Algorithm 2.6 is the calculation of d which requires a search of the surrounding area of the endpoint of the ray, $(x_{z^j_{g,t}}, y_{z^j_{g,t}})$, for the closest occupied cell. In order to avoid this pitfall, the likelihood field can actually be pre-computed and cached in a table lookup. Figure 2.7 shows an example

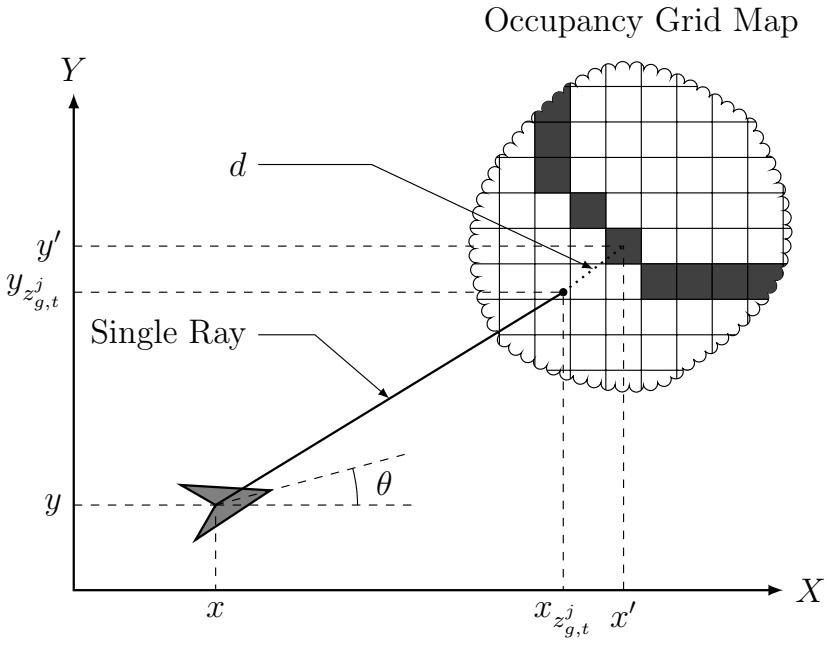
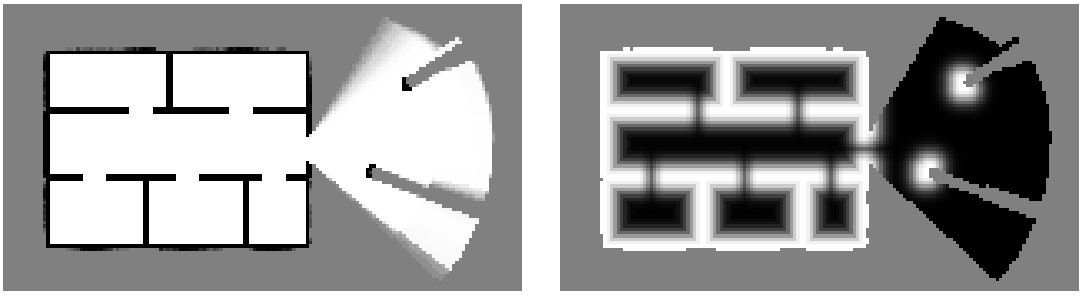


Figure 2.6: Illustration on how the Likelihood Field Range Finder Model is derived

of this cached look-up table turned into a grayscale image along with the occupancy grid map it was based on. Each cell in a likelihood field stores the probability of an beam endpoint being within that cell.



(a) Occupancy Grid (b) Pre-processed Likelihood Field

Figure 2.7: Example of a likelihood field lookup table.

Map Integration of Sensor Observations

The 'integrateSensorObs' function from Algorithm 2.3 also varies in implementation depending on the map being used. Algorithm 2.7 and Algorithm 2.8 show how the feature and LRF measurements are integrated into feature map and occupancy grid maps respectively. In Algorithm 2.7, $h^{-1}(z_{f,t}^{(i,j)}, x_t^{(i)})$ is based on (2.7) and defined as

$$h^{-1}(\mu, x_t) = \begin{bmatrix} \mu_x \\ \\ \mu_y \end{bmatrix} = \begin{bmatrix} x + r\cos(\phi + \theta) \\ \\ y + r\sin(\phi + \theta) \end{bmatrix} \tag{2.11}$$

where $(x, y, \theta)^T$ is the current robot pose.

Resampling Condition

Finally, the last aspect of the generalized FastSLAM algorithm Algorithm 2.3 that needs to be explored is the 'Resampling Condition'. One of the chief disadvantages of particle filters is that it is possible that it may suffer from particle depletion where it is impossible for the filter to provide correct state estimation because particles close to the true state estimate have been previously discarded. This issue could be solved by introducing random state estimates in particles, but this can be quite difficult and impossible in a mapping scenario. Therefore, it is better to instead try to prevent particle depletion in the first place.

One way of achieving this is by not resampling at every time step. The most popular resampling condition was developed by Doucet *et al.* in [11]. Doucet's method calculates what is known as the 'effective sample size' of N_{eff} of the particle filter

Algorithm 2.7 Integration of feature measurements in feature maps [5]

Require:

$m_{f,t-1}^{(i)} = \left\langle \langle \mu_{1,t-1}^{(i)}, \Sigma_{1,t-1}^{(i)} \rangle, \cdots, \langle \mu_{F,t-1}^{(i)}, \Sigma_{F,t-1}^{(i)} \rangle \right\rangle$, feature map from previous time step with F number of features

$x_t^i = (x, y, \theta)^T$, current proposed pose

$z_{f,t}$, most recent feature measurement

Q, measurement noise covariance matrix

h, measurement model

Ensure:

$m_{f,t}^{(i)}$, updated map

$m_{f,t}^{(i)} = \{\}$ // *Empty Map*

for all observed features j in $z_{f,t}$ **do**

  if feature j's first observation **then**

    $\mu_{j,t}^{(i)} = h^{-1}(z_{f,t}^{(i,j)}, x_t^{(i)})$ // *mean initialization*

    $H = h'(x_t^{(i)}, \mu_{j,t}^{(i)})$ // *compute Jacobian*

    $\Sigma_{j,t}^{(i)} = H^{-1} Q (H^{-1})^T$ // *covariance initialization*

  else

    $\hat{z}_{f,t}^{(i,j)} = h(\mu_{j,t-1}^{(i)}, x_t^{(i)})$ // *measurement prediction*

    $H = h'(x_t^{(i)}, \mu_{j,t-1}^{(i)})$ // *compute Jacobian*

    $Q_t = H \Sigma_{j,t-1}^{(i)} H^T + Q$ // *measurement covariance*

    $K = \Sigma_{j,t-1}^{(i)} H^T Q_t^{-1}$ // *Kalman gain*

    $\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K(z_{f,t}^{(j)} - \hat{z}_{f,t}^{(i,j)})$ // *update mean*

    $\Sigma_{j,t}^{(i)} = (I - KH)\Sigma_{j,t-1}^{(i)}$ // *update covariance*

  end if

  $m_{f,t}^{(i)} = m_{f,t}^{(i)} \cup \langle \mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)} \rangle$

end for

for features j in $m_{f,t-1}^{(i)}$ unobserved in $z_{f,t}$ **do**

  $\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)}$

  $\Sigma_{j,t}^{(i)} = \Sigma_{j,t-1}^{(i)}$

  $m_{f,t}^{(i)} = m_{f,t}^{(i)} \cup \langle \mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)} \rangle$

end for

Algorithm 2.8 Integration of LRF data in occupancy grid maps [5]

Require:
$m^{(i)}_{g,t-1} = \langle c^{(i)}_{1,t-1}, c^{(i)}_{2,t-1}, \cdots, c^{(i)}_{G,t-1} \rangle$, occupancy grid map from previous time step with G number of cells
$x^i_t = (x, y, \theta)^T$, current proposed pose
$z_{g,t}$, most recent LRF measurement
Ensure:
$m^{(i)}_{g,t}$, updated map

$m^{(i)}_{g,t} = m^{(i)}_{g,t-1}$ *// Copy existing map*
for all $c^{(i)}_{j,t-1}$ in $m^{(i)}_{g,t-1}$ **do**
  if $c^{(i)}_{j,t-1}$ in perceptual field of $z_{g,t}$ **then**
    $(x_j, y_j) =$ center of $c^{(i)}_{j,t-1}$
    $r = \sqrt{(x_j - x)^2 + (y_j - y)^2}$
    $\phi = \text{atan2}\,(y_j - y, x_j - x) - \theta$
    $k = \text{argmin}_\tau\, |\phi - \theta_{\tau,\text{sens}}|$
    if $r > \min\,(z_{\max}, z^k_{g,t} + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ **then**
      $\lambda = \lambda_0$
    else if $z^k_{g,t} < z_{\max}$ and $|r - z^k_{g,t}| < \alpha/2$ **then**
      $\lambda = \lambda_{\text{occ}}$
    else *// $r \leq z^k_{g,t}$*
      $\lambda = \lambda_{\text{free}}$
    end if
    $c^{(i)}_{j,t} = c^{(i)}_{j,t-1} + \lambda - \lambda_0$
  end if
end for

and is computed by the equation

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w^{(i)})^2} \tag{2.12}$$

From (2.12), it is clear that the lower the variance in the weights yields a higher N_{eff} value, and vice versa. A higher N_{eff} value means that the filter is equally confident in all the current particles and thus, resampling could result in the loss of particles closest to the true state. Conversely, a lower N_{eff} value signifies that a few particles have a much higher weight than the rest, thus the particle filter is confident in just a few particles. This leads to an opportune time for the filter to resample and get rid of lower weighted particles and keep the higher weighted and more likely particles. The threshold most commonly used by this method is to resample whenever $N_{eff} < N/2$ [1].

2.3 Reinforcement Learning

Reinforcement learning attempts to find a mapping from states S to actions A which maximimizes a numerical reward signal r [12]. The learning algorithm is not instructed as to which actions to take, but rather must determine which actions yield the most reward by trying them. Eventually, once enough interaction with the environment has been performed, the algorithm will converge towards a desired mapping, better known as a *policy*.

A great deal of research has been done in this field and produced a number of learning approaches. Different approaches work more favorably depending on the amount of prior knowledge of the environment. In the work done by Wurm *et al.* [1] and this thesis, the model of the environment is not known and thus Monte Carlo

methods or Temporal-Difference (TD) methods are possible options.

TD learning is a combination of Monte Carlo ideas, where learning is achieved directly from raw experience without a model of the environment's dynamics, and dynamic programming ideas, where estimates are updated based in part on other learned estimates. Wurm *et al.* opted for the use of the popular SARSA algorithm [12] which does not require a model of the environment. It learns an action-value function $Q(s, a)$ which contains a value for every state-action pair. The basic algorithm described in [12] is shown in Algorithm 2.9. α is known as the learning rate and γ is known as the discounting rate.

The selection of which action to perform is typically done through an ε-greedy policy. A greedy policy chooses the action a which has the highest value $Q(s, a)$ in state s. Whereas an ε-greedy policy will explore a non maximum random action with likelihood ε.

Algorithm 2.9 SARSA Algorithm [12]

Initialize $Q(s, a)$ arbitrarily
for all episodes **do**
  Initialize s
  Choose a from s using policy derived from Q
  repeat
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a';$
  until s is terminal
end for

2.3.1 Learning in Mapping

Wurm *et al.* developed a method capable of using reinforcement learning to take advantage of each map representation by having each particle contain a feature map

and grid map [1]. The basic principle of their work was to determine which method of pose sampling/propagation should be used at each time step. By selecting an appropriate model that considers the current state estimate, sensor observations, and odometry readings, the reinforcement learning algorithm can converge to a decision of which proposal distribution method should be used depending on the current model state. If grid-based is chosen then Algorithm 2.5 is used, and if feature-based is chosen then Algorithm 2.4 is used.

The model used by Wurm had an action set of $A = \{a_g, a_f\}$, where a_g defined the use of the grid-based proposal and a_f defined the use of the feature-based proposal. The state set, S, was defined so that it represented all the necessary information from sensor observation and particle filter effective sample size values to best make a decision. Their state consisted of an average scan matching likelihood $\bar{l}$, a boolean variable given by $N_{eff}^{f} < N_{eff}^{g}$, and a boolean variable indicating if a feature was detected in the current time step, resulting in

$$S := \{\bar{l}\} \times \{1_{N_{eff}^{f} < N_{eff}^{g}}\} \times \{1_{\text{feature detected}}\} \tag{2.13}$$

The value of $\bar{l}$ was discretized into seven different intervals $(0.0-0.15, 0.16-0.3, 0.31-0.45, 0.46-0.6, 0.61-0.75, 0.76-0.9, 0.91-1.0)$. Thus, the total number of states were $7 \times 2 \times 2 = 28$.

The average scan matching likelihood $\bar{l}$ variable is calculated using

$$\bar{l} = \frac{1}{N} \sum_{i=1}^{N} (\max_{x_t} p(z_{g,t} | x_t^{(i)}, m_{g,t}^{(i)})) \tag{2.14}$$

where x_t is the pose that maximizes the probability value $p(z_{g,t} | x_t^{(i)}, m_{g,t}^{(i)})$. To evaluate this value, the aforementioned 'beam endpoint model' and Algorithm 2.6 are used.

Wurm *et al.* performed the training using simulated data in order to have access to the true robot pose x_t^* at every time step t. Having access to the true pose also allows for the reward function to be the weighted average deviation from the true pose. In an effort to not punish a current action for a previous decision, only the deviation accumulated since the previous step $t-1$ is used:

$$r(s_t) = \sum_{i=1}^{N} w_{t-1}^{(i)} ||x_{t-1}^{(i)} - x_{t-1}^*|| - \sum_{i=1}^{N} w_t^{(i)} ||x_t^{(i)} - x_t^*|| \tag{2.15}$$

where $w_{t-1}^{(i)}$ and $w_t^{(i)}$ vary depending on the action chosen since each particle contains two different sets of weights. At time t, $w_{g,t}^{(i)}$ is used if a_g was chosen and $w_{f,t}^{(i)}$ is used if a_f was taken. Once this learning is complete, the derived policy can be used in conjunction with Wurm's overall mapping algorithm as listed in Algorithm 2.10. This algorithm is used in this thesis as the method used to map the indoor and outdoor environments. It was required to both implement, train, and validate this algorithm before being able to use it.

2.4 Map Merging

This component of mapping is critical to multi-robot SLAM and has been receiving more attention in the literature. The easiest way to map increasingly larger environments efficiently is to involve multiple mapping robots. A centralized approach to mapping might only merge the map once the entire environment has been explored [13]. Conversely, a decentralized approach would merge maps once individual robots meet each other in the environment [14]. Regardless of the approach, the act of merging requires the calculation of an appropriate transformation matrix from one robot global reference frame to the other robot's global frame of reference. This

Algorithm 2.10 Wurm's Combined Mapping Approarch [1]

Require:
S_{t-1}, the sample set from the previous time step
$z_{g,t}$, the most recent LRF measurement
$z_{f,t}$, the most recent feature measurement
u_{t-1}, the most recent odometry measurement
Ensure:
S_t, the new sample set

maptype = decide$(S_{t-1}, z_{g,t}, z_{f,t}, u_{t-1})$

$S_t = \{\}$
for all $s_{t-1}^{(i)} \in S_{t-1}$ **do**
$\langle x_{t-1}^{(i)}, w_{g,t-1}^{(i)}, w_{f,t-1}^{(i)}, m_{g,t-1}^{(i)}, m_{f,t-1}^{(i)} \rangle = s_{t-1}^{(i)}$

if (maptype=grid) **then** // *Compute proposal*
$x_t^{(i)} \sim p(x_t | x_{t-1}^{(i)}, u_{t-1}, z_{g,t})$
else
$x_t^{(i)} \sim p(x_t | x_{t-1}^{(i)}, u_{t-1})$
end if

$w_{g,t}^{(i)}$ =updateGridWeights$(w_{g,t-1}^{(i)}, m_{g,t-1}^{(i)}, z_{g,t})$
$w_{f,t}^{(i)}$ =updateFeatureWeights$(w_{f,t-1}^{(i)}, m_{f,t-1}^{(i)}, z_{f,t})$
$m_{g,t}^{(i)}$ =integrateScan$(m_{g,t-1}^{(i)}, x_t^{(i)}, z_{g,t})$
$m_{f,t}^{(i)}$ =integrateFeatures$(m_{f,t-1}^{(i)}, x_t^{(i)}, z_{f,t})$

$S_t = S_t \cup \{\langle x_t^{(i)}, w_{g,t}^{(i)}, w_{f,t}^{(i)}, m_{g,t}^{(i)}, m_{f,t}^{(i)} \rangle\}$ // *update sample set*
end for
for $i = 1 \rightarrow N$ **do**
if (maptype=grid) **then**
$w_t^{(i)} = w_{g,t}^{(i)}$
else
$w_t^{(i)} = w_{f,t}^{(i)}$
end if
end for

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^{(i)})}$$

if $N_{eff} < N/2$ **then**
S_t =resample$(S_t, \{w_t^{(i)}\})$
end if

transformation matrix will take the form of:

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & t_x \\ & T_\theta & \\ & & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

where θ is the rotation parameter between the reference frames and t_x, t_y are the translation parameters.

Since this thesis focuses on a decentralized map merging algorithm that is from the perspective of one robot, the *current* robot will be the one who is merging the *other* robot's maps into its own.

Ozkucur and Akin devised a method of merging feature maps [14] which includes first calculating the mean of the feature parameters across all particles for the other robot. The mean is then transformed and merged with all the current robot's particles using the following equations:

$$p_m = p_c + \Sigma_c[\Sigma_c + T_\theta^T \Sigma_o T_\theta]^{-1}(p_T - p_c) \tag{2.17}$$

$$\Sigma_m = \Sigma_c - \Sigma_c[\Sigma_c + T_\theta^T \Sigma_o T_\theta]^{-1}\Sigma_c \tag{2.18}$$

where p_T is the transformed coordinates extracted from the calcuation $[x_T, y_T, 1]^T = T[x_o, y_o, 1]^T$; p_m, p_c, and p_o are the merged, current robot's estimate of, and other robot's estimate of feature position respectively; and Σ_m, Σ_c, and Σ_o are the merged covariance, current robot's covariance, and other robot's covariance for the feature position. Figure 2.8 shows an example of how another robot's feature map is transformed and then merged with the current robot's map to produce a final feature map.

Notice how the ellipses representing covariance actually shrink after the merge. Logically, this makes sense since the covariance normally shrinks after each observation and the merging of covariance matrices is similar to adding several observations at once.

An occupancy grid map is an array of cells similar to how an image is an array of pixels. Therefore, they can also be transformed and then merged by using (2.16). Once an incoming cell position has been transformed and its destination cell determined, the values simply need to be added since each cells value is a log likelihood. Figure 2.9 shows an example of how another robot's occupancy grid map is transformed and then merged with the current robot's map to produce a final occupancy grid map.

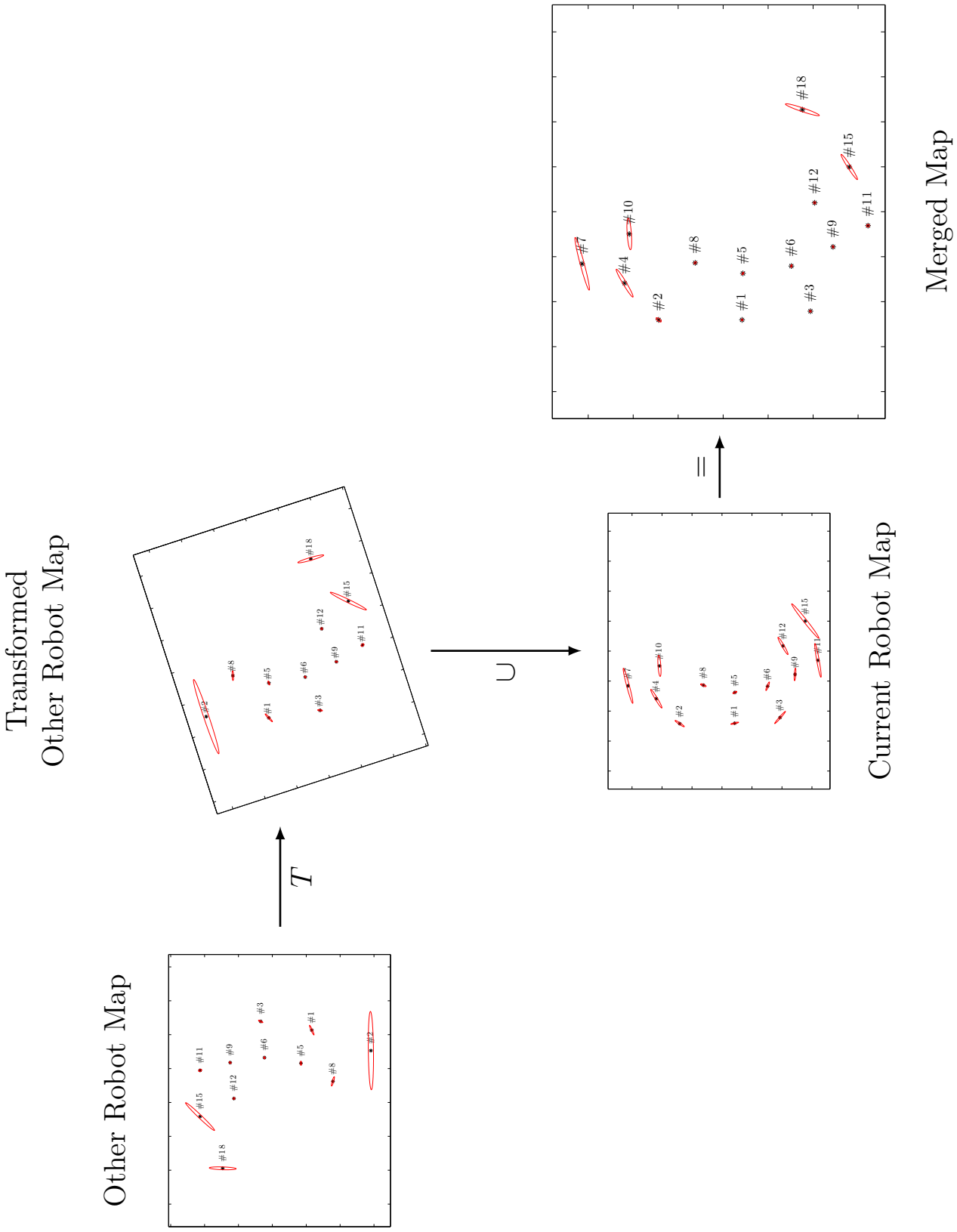


Figure 2.8: Flowchart showing steps to merge feature maps

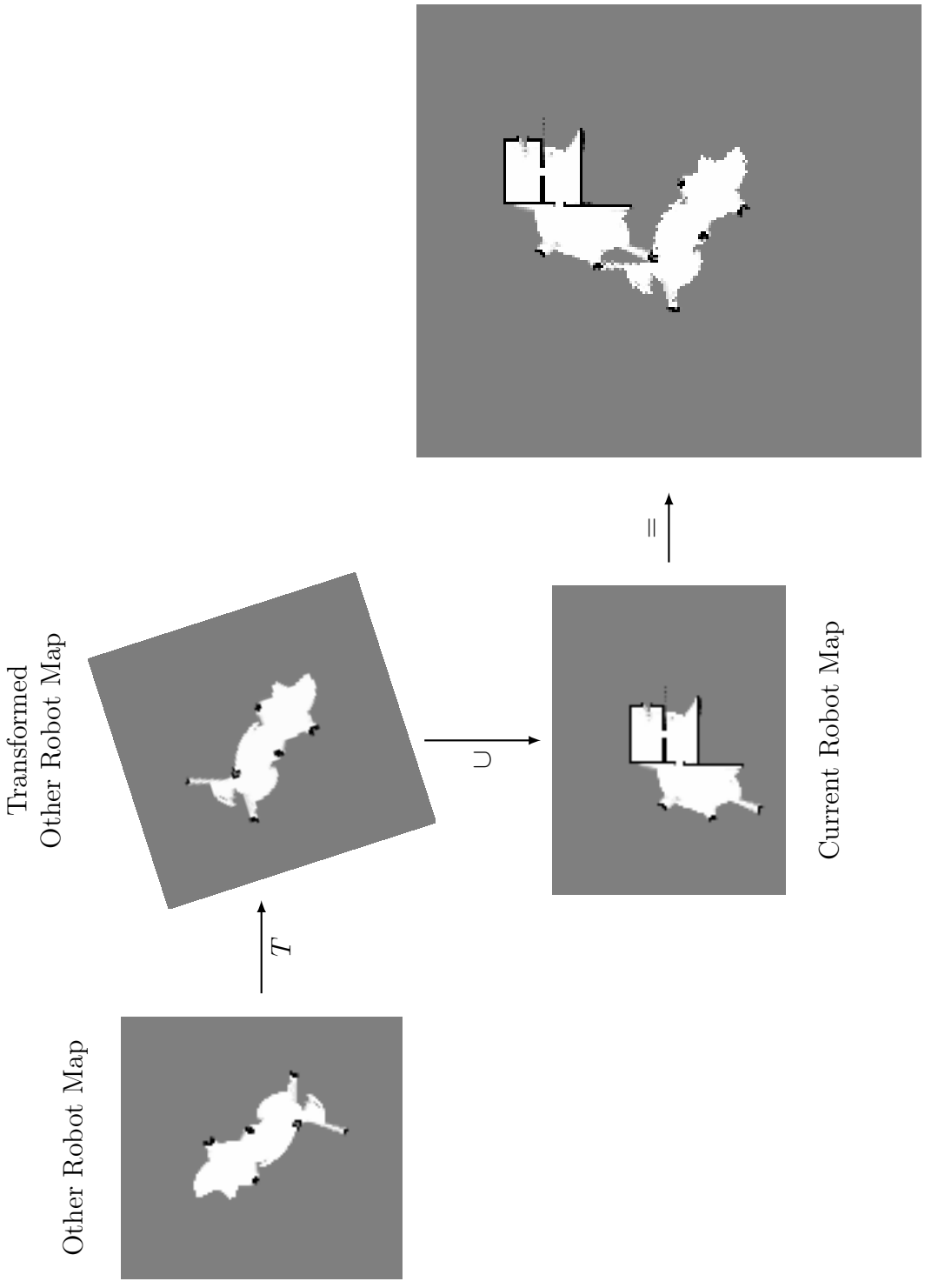


Figure 2.9: Flowchart showing steps to merge occupancy grid maps

Chapter 3

Software Resources Development

There exists multiple software packages such as 'Player/Stage' [15], 'Microsoft Robotics Studio' [16], and 'CARMEN' [17] that are capable of generating simulated datasets. However, these packages only take images as input to represent maps of an environment. This means that the accuracy of the simulated sensors are limited to the real world size represented by each image pixel. Also, the environment would be quite 'square' due to the inherent shape of a pixel image. Therefore, a custom simulation engine would need to be created to achieve more realistic mobile robot datasets. Instead of developing custom software to process these datasets, an existing software package, known as the Mobile Robotics Programming Toolkit (MRPT) [18], was found that included the necessary mapping algorithms and only required some extensions.

3.1 Mobile Robot Data Simulation Engine

To reduce complexity, the simulation engine would assume that an environment was only two dimensional and thus only having three degrees of freedom. This assumption is proper due to the fact that all maps used in this thesis are also two dimensional.

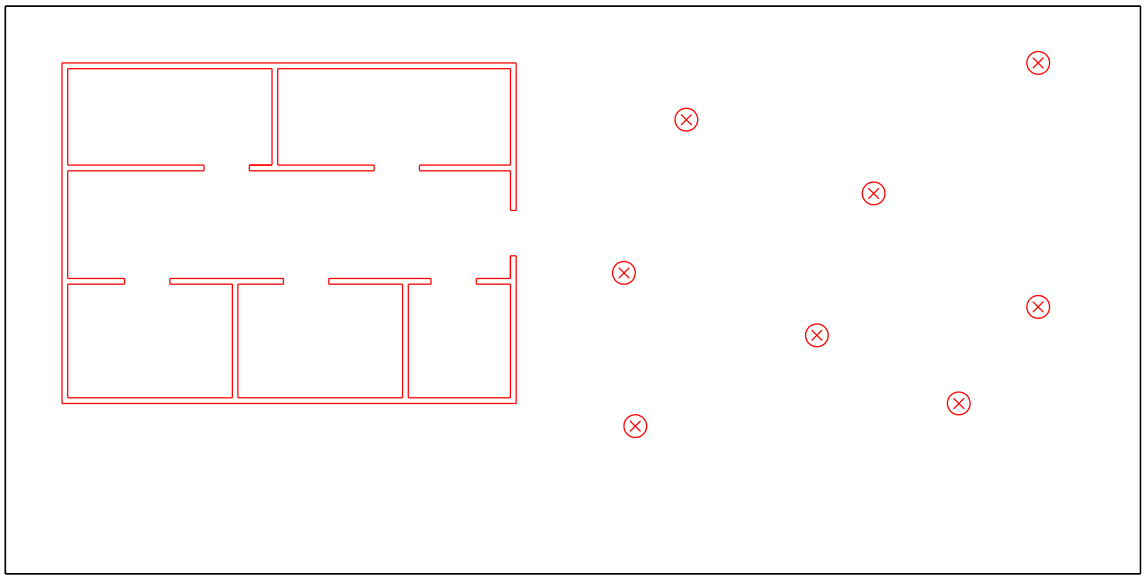


Figure 3.1: Simple Indoor and Outdoor Environment

The entire engine was developed in MATLAB.

3.1.1 Creation of Environments

To remain simple, an indoor and outdoor environment was designed. A structured indoor environment is represented by lines and an outdoor environment is represented by circles. Circles were chosen to represent features due to the fact that many outdoor feature based mapping methods will use trees and posts with circular cross-sections as features. A sample environment drawn in AutoCAD is shown in Figure 3.1.

Although the method used to produce environments is independent of the simulation engine, for this thesis AutoCAD was chosen to produce environments. For environment input, the 'building' lines need to be represented by the two endpoints and the 'features' need to be represented by the centre of the circle and by the radius.

3.1.2 Simulated Sensors

In order to properly produce realistic datasets, it is necessary to accurately simulate typical real world sensors. Another requirement for these simulated sensors is that they should be as customizable as possible.

Encoders

A wheel encoder can be installed on a vehicle in order to measure the wheel rotations, which can in turn be used to measure displacement, $d_k + n_{d,k}$, and speed, $u_k + n_{v,k}$, for a wheel. Noise in each measurement is represented by $n_{d,k}$ and $n_{v,k}$ for displacement d_k and speed u_k respectively. To easily simulate this sensor, the true wheel speed of the simulated robot can have reasonably chosen Gaussian white noise added to it to produce a realistic wheel encoder reading.

To determine the variance of the white noise, parameters to represent the 'simulated' wheel encoder must be chosen. The encoder being simulated is assumed to have a bit count of $N = 8$ and the vehicle has wheel with a radius of $r = 0.1$ m. Given the previous parameters, the resolution of the sensor is:

$$\frac{2\pi r}{2^N} = \frac{0.2\pi}{256}\text{m}. \tag{3.1}$$

Thus, the quantization error of displacement is

$$\pm\frac{1}{2} \times \frac{0.2\pi}{256} = \pm\frac{0.1\pi}{256}\text{m} \tag{3.2}$$

on each measurement. Even though the noise in an encoder is better represented by a uniform distribution, it is being assumed that the quantization error is normally

distributed such that $n_{d,k} \sim \mathcal{N}(0, \sigma_d^2)$ with

$$3\sigma_d = \frac{0.1\pi}{256}\text{m}. \tag{3.3}$$

An estimate of a vehicle's speed can be achieved using a first-order backwards-difference approximation:

$$u_k \approx \frac{1}{T}(d_k - d_{k-1}) \tag{3.4}$$

where T is the duration of one time step. The time step used in this simulation was $T = 0.1$ sec. Assuming that the error is uncorrelated between measurements and pass those measurements through the first-order differentiator, then the result is

$$\text{speed} = u_k + n_{v,k} \tag{3.5}$$

where $n_{v,k} \sim \mathcal{N}(0, \sigma_v = \frac{2}{T^2}\sigma_d^2)$. Therefore, in order to 'simulate' a wheel encoder, white noise with standard deviation

$$\sigma_v = \sqrt{\frac{2}{T^2}\left(\frac{0.1\pi}{3 \cdot 256}\right)^2} \approx 0.005785 \tag{3.6}$$

must be added to the true wheel speeds.

Laser Range Finder

A Laser Range Finder (LRF) is an active sensor that projects laser rays outwards into the environment, these rays bounce off objects in the environment then return to the sensor producing a two dimensional cross section of the surronding environment. To simulate this sensor, first each ray must be projected until it hits an object, line or cicle, in the environment. If there are no objects for the ray to hit or the object is outside of the range of the LRF, then the maximum range distance of the LRF is

returned. Otherwise, the distance travelled by the ray to the object is returned as the measurement.

To be realistic, noise must be added to this true measurement. Accurate noise models for LRFs take into account multiple components. First, in a dynamic environment there is a probability that a ray might hit a moving object as opposed to the static object it would have otherwise. Second, there is a probability that the ray will be reflected away from the sensor causing a maximum length reading. Third, there is a probability of a uniformly distributed random reading due to unknown sensor error. Finally, there is some Gaussian error in a properly propagated ray reading [5]. However, for the purposes of this thesis, the noise model was simplified to only have error in a properly propagated ray reading. This simplication comes from the assumption that the environment is static, the environment properly reflects the rays, and that the simulated sensor does not produce random errors that a real sensor would. In practice, these assumptions should not be made but in this case they should not affect the results presented in this thesis.

Therefore, each ray, i, returns a distance, $z_g^i + n_{\text{LRF}}$, where $n_{\text{LRF}} \sim \mathcal{N}(0, \sigma^2_{\text{LRF}})$. The value of the parameters for the simulate LRF are approximately based on the low cost Hokuyo URG-04LX-UG01 and listed in Table 3.1.

Figure 3.2 shows an example of a LRF measurement without noise and the same reading with exagerated noise, i.e. being shown not to scale.

Feature Detector

Feature maps require that features need to be extracted from available sensor data. A feature detector measurement can have both the distance, i.e. range, to a feature and its relative position, i.e. bearing, or it can simply have one or the other. The

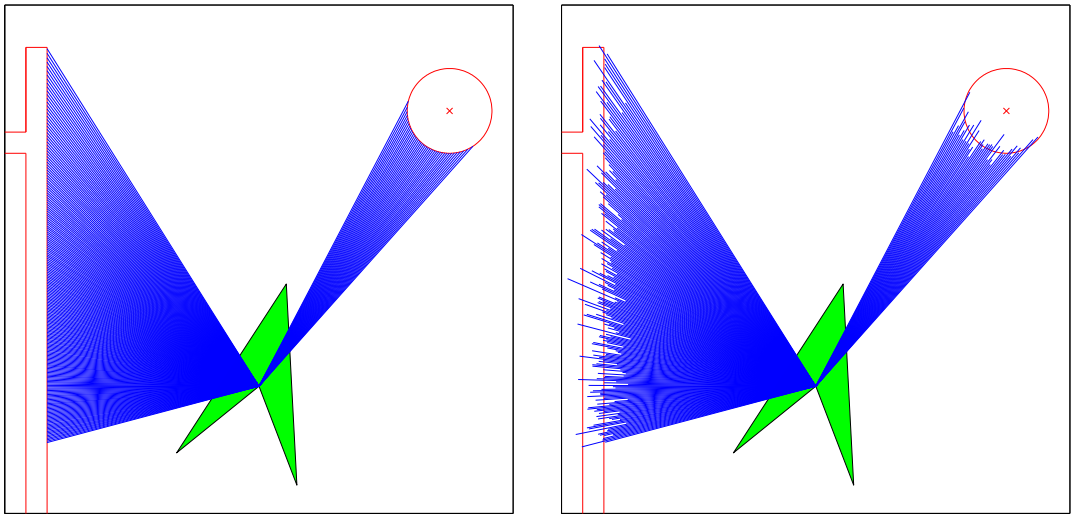
Parameter	Value
σ_{LRF}	0.015 m
Field of View	240°
Angular Resolution	0.3516°
Minimum Length	0.1 m
Maximum Length	5.0 m
Number of Rays	683
Distance Resolution	0.001 m

Table 3.1: Simulated Laser Range Finder Parameters

feature detector being simulated outputs both the range and bearing.

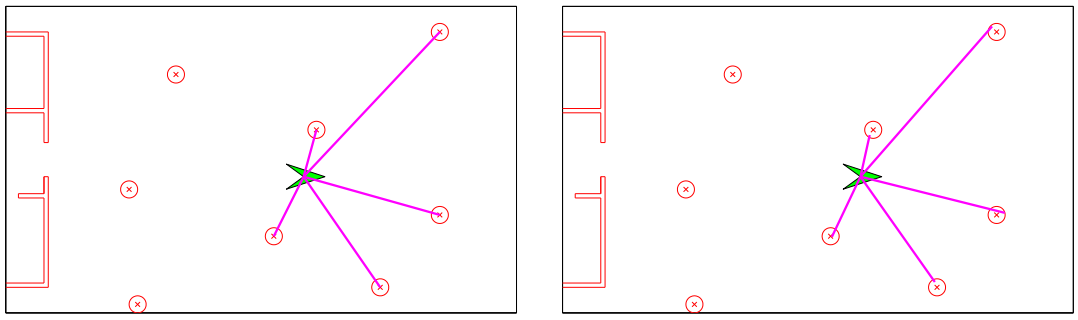
One common method to retrieve feature measurements is to label objects in the environment with some type of ID tag. A camera on the vehicle is used to identify and locate the ID. Another method is to use a long range LRF that covers a large area, from this data clusters of beam end points that are surrounded by free space can be extracted as a feature. For the purposes of this thesis, it is being assumed that a feature extractor already exists that is capable of providing range and bearing measurements for each feature in the environment. It is also assumed that each feature in the environment can be identified by the feature extractor.

Similar to the encoders, to simulate realistic sensor readings, the true measurements are used and then Gaussian white noise is added. Table 3.2 lists the value of the parameters used in the simulated feature detector and Figure 3.3 shows an example of actual range and bearing measurements next to a corrupted version. This simulated sensor has the ability to accurately identify each individual feature.



(a) LRF Scan Truth (b) LRF Scan with Exagerated Error

Figure 3.2: Sample LRF scan measurement



(a) Feature Reading Truth (b) Feature Reading with Exagerated Error

Figure 3.3: Sample feature detector measurement

Parameter	Value
σ_{range}	0.25 m
σ_{bearing}	0.5°
Field of View	240°
Maximum Length	15.0 m
Distance Resolution	0.001 m

Table 3.2: Simulated Feature Detector Parameters

Other Robot Sensors

Finally, there needs to be a sensor capable of detecting another robot in the neighbouring area. In multi-robot research, typically robots will have some type of id to allow other robots to recognize them. Since another robot in the environment is simply a moving 'feature', the sensor that measures the other robot relative position can be simulated similarly to the feature detector. Also, it is being assumed that each robot can be easily identified once observed.

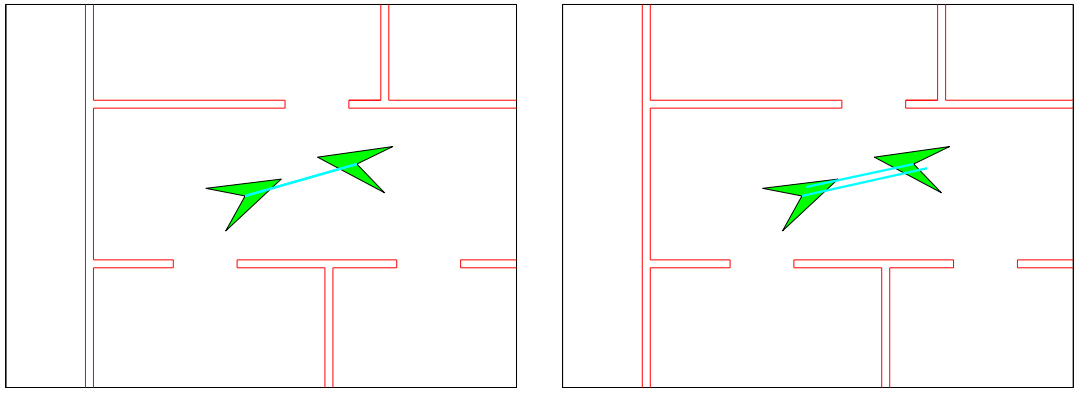
The reading from this sensor provides the range and bearing to the other robot. Again, the simulated sensor uses the actual measurements and then add Gaussian white noise. Table 3.3 details the parameters of the simulated sensor and Figure 3.4 demonstates the measurement reading, without and with error, when two robots mutually identify the other. This simulated sensor also has the ability to accurately identify other robots.

3.1.3 Trajectory Tracking

To produce realistic datasets, the simulated mobile robot must traverse the environment in realistic movements. In this thesis we simulate a mobile robot that has a

Parameter	Value
σ_{range}	0.25 m
σ_{bearing}	0.5°
Field of View	240°
Maximum Length	15.0 m
Distance Resolution	0.001 m

Table 3.3: Simulated Other Robot Parameters



(a) Other Robot Reading Truth

(b) Other Robot Reading with Exagerated Error

Figure 3.4: Sample other robot measurement

differential drive wheel base as illustrated in Figure 3.5. Using a body-centered axis model and assuming that the wheels do not slip laterally, the vehicle model can be derived as:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \upsilon_R \\ \upsilon_L \end{bmatrix} \tag{3.7}$$

where $\mathbf{q} = (x, y, \theta)^T \in \mathbb{R}^2 \times \mathbb{S}^1$ represents the pose, θ is the direction of the robot, l is the distance between wheels, $(\upsilon_R, \upsilon_L)^T \in \mathbb{R}$ are the input linear wheel speeds. All

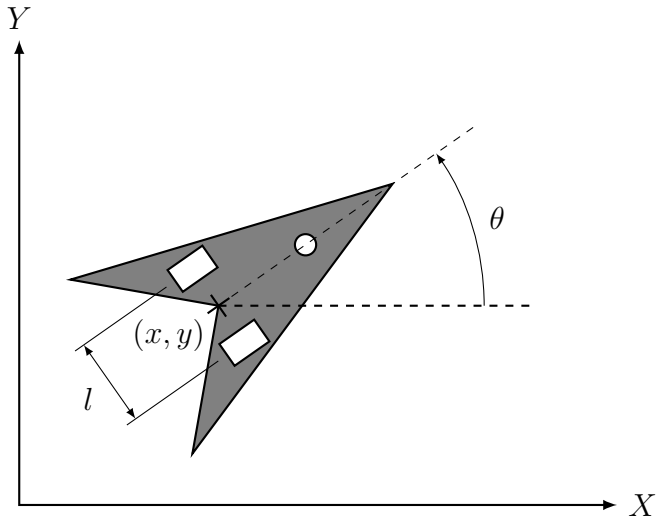


Figure 3.5: Diagram showing the state variables of a differential drive vehicle

derivations of equations for this section are shown in Appendix A.

It is easier to design a controller for a simpler, unicycle, vehicle model and convert those input signals for the differential drive model. The unicycle model is:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \upsilon \\ \omega \end{bmatrix} \tag{3.8}$$

where $\omega \in \mathbb{R}$ is the steering speed or rotational rate and $\upsilon \in \mathbb{R}$ is the linear vehicle speed.

The conversion matrix necessary to convert the inputs of the unicycle model to

the differential drive model is:

$$\begin{bmatrix} \upsilon_R \\ \\ \upsilon_L \end{bmatrix} = \begin{bmatrix} 1 & \frac{l}{2} \\ \\ 1 & -\frac{l}{2} \end{bmatrix} \begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} \tag{3.9}$$

One method of successfully tracking a trajectory represented by $z_d = (x_d, y_d)^T$ is to use exact linearization by dynamic extension [19]. As opposed to using only state feedback from $z = (x, y)^T$, this method uses additional dynamic states. In trying to find new states, time derivatives of z are taken. As shown in Appendix A, additional states are eventually found once an invertible matrix is reached. These new states are:

$$\begin{bmatrix} \dot{\upsilon} \\ \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ \\ -\frac{\sin\theta}{\upsilon} & \frac{\cos\theta}{\upsilon} \end{bmatrix} \begin{bmatrix} \eta_1 \\ \\ \eta_2 \end{bmatrix} \tag{3.10}$$

where $\eta := \ddot{z} = (\ddot{x}, \ddot{y})^T$ and is only valid when $\upsilon \neq 0$. The variable η will be used as the control input acceleration into the linearized system.

This dynamic extension allows for linearization through the selection of new coordinates $(\zeta_1, \zeta_2, \zeta_3, \zeta_4)^T := (x, y, \dot{x}, \dot{y})^T$ and produces the following feedback linearized system

$$\begin{bmatrix} \dot{\zeta}_1 \\ \\ \dot{\zeta}_2 \\ \\ \dot{\zeta}_3 \\ \\ \dot{\zeta}_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ \\ 0 & 0 & 0 & 1 \\ \\ 0 & 0 & 0 & 0 \\ \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \begin{bmatrix} \zeta_1 \\ \\ \zeta_2 \\ \\ \zeta_3 \\ \\ \zeta_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ \\ 0 & 0 \\ \\ 1 & 0 \\ \\ 0 & 1 \end{bmatrix}}_{B} \begin{bmatrix} \eta_1 \\ \\ \eta_2 \end{bmatrix} \tag{3.11}$$

Given that (3.11) is the linear system $\dot{\zeta} = A\zeta + B\eta$, full state feedback can be used

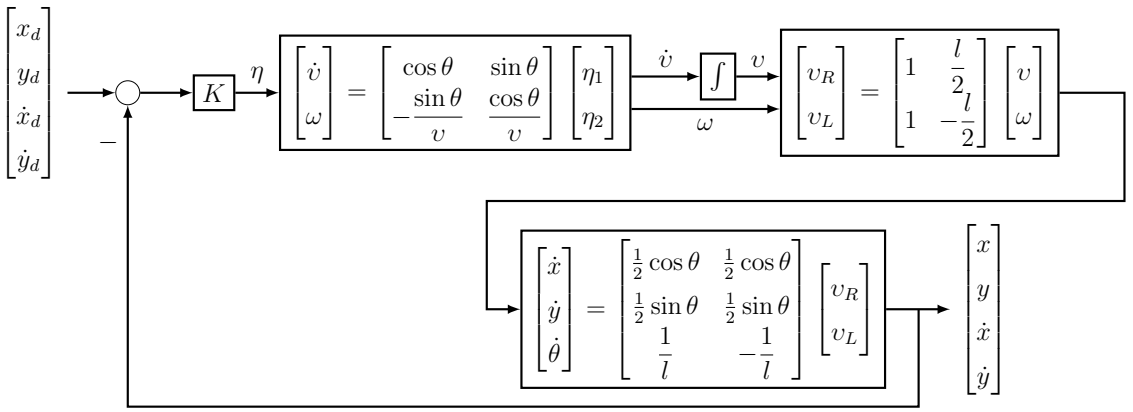


Figure 3.6: Controller Architecture for Trajectory Tracking

along with pole placement to produce a control signal capable of $\lim_{t\to 0}(\zeta_d - \zeta) = 0$ when $\eta = K(\zeta_d - \zeta)$. A pole placement technique was used to placed the closed loop poles at $(-1, -2, -2.5, -1.5)$ which yields the following control gains:

$$K = \begin{bmatrix} 3.75 & 0 & 4 & 0 \\ 0 & 2 & 0 & 3 \end{bmatrix} \tag{3.12}$$

Figure 3.6 shows the whole trajectory tracking controller architecture.

To implement this controller architecture (3.7) and (3.11) must be discretized. Since (3.7) is a non-linear system, it must be assumed that the state and input remain constant between samples in order for it to become:

$$\mathbf{q_k} = \mathbf{q_{k-1}} + T \cdot \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} v_R \\ v_L \end{bmatrix} \tag{3.13}$$

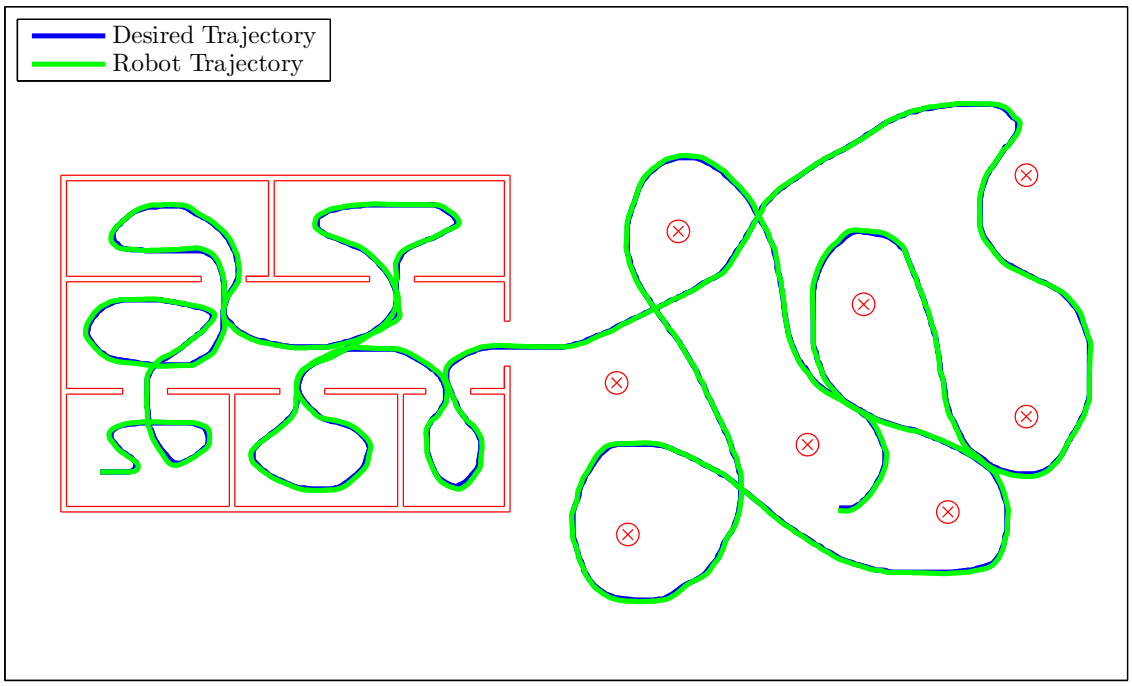


Figure 3.7: Trajectory Tracking Result

Conversely, (3.11) is linear and can be discretized using

$$\zeta_k = e^{(A-BK)T}\zeta_{k-1} \tag{3.14}$$

Desired trajectories are nothing more than a series of waypoints, that are then broken down into smaller waypoints that the robot can realistically reach during each timestep. Figure 3.7 shows the successful result of a trajectory being tracked by the described controller architecture.

3.2 MRPT modifications

After a few trials at implementing algorithms described in chapter 2 in MATLAB, it was determined that MATLAB would not meet the computational speed requirements

for this thesis. The Mobile Robot Programming Toolkit is a open source C++ software package with several SLAM algorithms already implemented by SLAM researchers. The MRPT toolkit met the computational speed requirements but lacked certain necessary features.

3.2.1 2D Feature Map Type

MRPT contained different map implementations but lacked a simple two dimensional feature map. As previously described in section 2.2.1, a feature map is composed of a vector of feature mean location coordinates and respective covariances. Thus in order to implement a feature map, two classes must be created.

The C2DFeature class represents individual features and its abbreviated header file is shown in section B.1.

The collection of features is handled by the C2DFeatureMap class and its abbreviated header file is listed in section B.2.

All MRPT map types can integrate sensor measurements through types having the CObservation base class. There was no need to implement a custom CObservation class, as the MRPT library already had a suitable observation type, CObservationBearingRange, that met the feature map compatibility requirements. There was also no need to implement an occupancy grid map class because MRPT already contained an extensive implementation named COccupancyGridMap2D.

3.2.2 Map Merging

The MRPT software package was not designed for multi-robot usage and thus did not contain any implementations of map merging algorithms. Therefore, both algorithms described in section 2.4 would need to be implemented. The implemented code for

merging two feature maps is shown in section B.3.

The contributors to MRPT provided the software package with an extensive matrix arithmetic library that facilitated the implementation of merging two feature maps. As shown in section B.3, if a feature is not found in the destination map, the transformed mean and covariance are simply added to the vector of features.

The implemented code used to accomplish a merging of two occupancy grids is listed in section B.4.

The first step of the implementation is to resize the current robot's map to be able to incorporate the other robot's map size and orientation. Resizing the map involves modifying the dimensions of the occupancy grid itself. This is to ensure that the destination cell of a transformed cell from the other robot's map does not fall outside the boundaries of the map of the current robot.

Once this is done, the other robot's map's pixels can be iterated through, transformed, and merged appropriately.

3.2.3 Other Robot Observation Type

As previously mentioned MRPT was not designed for a multi-robot implementation, thus a new observation class needed to be created. Since there are many similarities between feature observations and other robot observations, it was appropriate to base this new class on the CObservationBearingRange. The abbreviated header file for the newly developed CObservationOtherRobot class is shown in section B.5.

For this thesis, all environments are assumed to be two-dimensional, thus the inclusion of a 'pitch' angle is to maintain consistency with the CObservationBearingRange class.

3.2.4 DataSet to Rawlog Converter

The dataset produced by the developed simulation engine is saved in MATLAB arrays and cell arrays and are not compatible with the MRPT library, which requires datasets in the form of its Rawlog type. Thus a program was developed that could convert multiple comma separated values (CSV) files into a Rawlog file. Each corresponding line in the CSV files would represent the sensor values at a particular time step.

The first step was to develop code in MATLAB that could output the arrays storing the sensor values into CSV files. The array of LRF readings was outputted verbatim with each line having the distance values for each ray. The output for feature and other robot measurements needed to be slightly different. The first number in a line would represent the number of features or robots that were observed. If any were observed, then the following pair values would represent the range and bearing.

The MRPT Rawlog format stores robot actions as either odometry with accompanying linear and angular velocities or encoder ticks. It was deemed easier to convert the wheel encoder speeds into odometric readings compatible with MRPT's Rawlog format. Assuming that the origin is the pose of a robot at t_0, the values of the simulated wheel encoders can be used as input into the discrete time vehicle model (3.13) to get the corresponding pose. The linear and angular velocities can be calculated from the inverse of (3.15) and is shown here:

$$\begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \upsilon_R \\ \\ \upsilon_L \end{bmatrix} \tag{3.15}$$

Another CSV file is produced containing the parameters for each sensor. Corresponding CSV files for all sensors containing the truth must also be produced. CSV files are then parsed and the data is inserted into instances of the appropriate class

and then inserted into the appropriate container class for MRPT's Rawlog format. Table 3.4 shows which classes must be used for each sensor reading.

Sensor	MRPT Class
Wheel Encoders	CActionRobotMovement2D
Laser Range Finder	CObservation2DRangeScan
Feature Extractor	CObservationBearingRange
Other Robot	CObservationOtherRobot

Table 3.4: Sensor to MRPT class mapping

3.2.5 Particle Filter Modifications

Some small modifications were made to the MRPT implementation of RBPF mapping. With the ubiquity of multi-core CPUs, it would be beneficial for any computationally intensive algorithm to take advantage of this. In particle filters, each particle is manipulated independent of any other particle. Typically this is done in some type of 'for loop'. Most modern C++ compilers have access to an API named Open Multi-Processing (OpenMP) that provides facilities to easily parallelize for loops. The following code block shows a simple example.

```
#pragma omp parallel for
for (int i=0; i < nParticleCount; i++)
{
  CParticle* pPart = arrParticles[i];

  /* perform necessary code on individual particles */

  #pragma omp critical
  /* perform any necessary code that it performed
     on an object used by all loops */
}
```

Once the work of Wurm *et al.* [1] was successfully replicated, it was necessary to adjust the implementation of RBPF in MRPT to properly follow Algorithm 2.10. Additionally, the determination of the current reinforcement learning state also needed to be implemented and the appropriate action for each state permutation was also stored.

Chapter 4

Proposed Approach

With the necessary foundation work outlined, the main contribution of this thesis can now be described. To have reinforcement learning reach a proper decision algorithm, the model must be appropriately selected and the reward determination properly designed. But first, any merging of maps requires the determination of a transformation matrix from other robot's global reference frame to the current robot's global reference, as mentioned in section 2.4. Given the available sensors, there are a couple of different ways of determining this matrix. As mentioned in section 3.1.2, all simulated sensors are responsible for the input of noise.

4.1 Calculation of Transformation Matrix

Each robot contains multiple particles each containing different maps, thus there are potentially $N_c \times N_o$ possible combinations of merged maps. The memory requirements necessary to store all of these maps is not feasible and some other method of keeping the number of resultant merged maps at a reasonable number must be devised. Ozkucur and Akin in [14] only dealt with the merging of feature based maps which allowed them to average the other robot's estimated poses and maps. This

method of averaging does not yield favorable results in occupancy grid maps due to their inherent image-like nature. Therefore, for the approach taken in this thesis the maps and pose estimates of the particle with the greatest weight will be 'sent' to the current robot for merging.

It is being assumed that a robot can only merge maps with another robot when they are in a rendezvous scenario where both mutually observe the other. To calculate the desired transformation matrix T, three different component transformation must be used. The first component, T_c, is the transformation matrix to convert points in the current robot's local frame of reference to its global frame of reference from its pose at t_0. This matrix is built using the pose of the current robot $x_c^{(i)} = (x_c, y_c, \theta_c)^T$ and there are N_c different poses, thus this means that there will be N_c different T matrices, i.e. one for each particle. The second component, T_r, is the transformation matrix to convert points in the other robot's local frame of reference to the current robot's local frame of reference. This matrix is built using the readings from Other Robot sensor. The final component, T_o, is the transformation matrix to convert points in the other's robot's frame of reference to its global frame of reference from its pose at t_0. This matrix is built using the pose 'sent' from the other robot $x_o^\dagger = (x_o, y_o, \theta_o)^T$. The following equation shows the calculation of T necessary to transform the map 'sent' from the other robot and merge into each of its particle's map:

$$T^{(i)} = T_c T_r T_o^{-1} =$$

$$\begin{bmatrix} \mathrm{c}\,\theta_c^{(i)} & -\mathrm{s}\,\theta_c^{(i)} & t_{x_c}^{(i)} \\ \mathrm{s}\,\theta_c^{(i)} & \mathrm{c}\,\theta_c^{(i)} & t_{y_c}^{(i)} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{c}\,\theta_r & -\mathrm{s}\,\theta_r & t_{x_r} \\ \mathrm{s}\,\theta_r & \mathrm{c}\,\theta_r & t_{y_r} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{c}\,\theta_o & -\mathrm{s}\,\theta_o & t_{x_o} \\ \mathrm{s}\,\theta_o & \mathrm{c}\,\theta_o & t_{y_o} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \tag{4.1}$$

where $t_{x_c} = x_c$, $t_{y_c} = y_c$, $t_{x_o} = x_o$, $t_{y_o} = y_o$, and s and c are shortform for sin and cos respectively. The relative transformation matrix terms t_{x_r}, t_{y_r}, and θ_r can be calculated using the range and bearing components from the mutual robot observations using the following equations:

$$\begin{aligned} t_{x_r} &= r\cos\theta_{c\to o} \\ t_{y_r} &= r\sin\theta_{c\to o} \\ \theta_r &= \pi - \theta_{o\to c} + \theta_{c\to o}. \end{aligned} \tag{4.2}$$

where $\theta_{c\to o}$ and $\theta_{o\to c}$ are illustrated in Figure 4.1.

Figure 4.1 shows how these equations were derived analytically and Figure 4.2 shows how all the component transformation matrices combine to form the desired transformation matrix.

The transformation matrix estimate $T^{(i)}$ is a first estimate and can be possibly improved in various ways in order to reduce the overlap inconsistencies that could occur upon merging. However, only one option for each map type will be used for this thesis. Since the robots are relatively close together when they meet, this means that the observations of the other robot at the time of merging can be used and fitted into

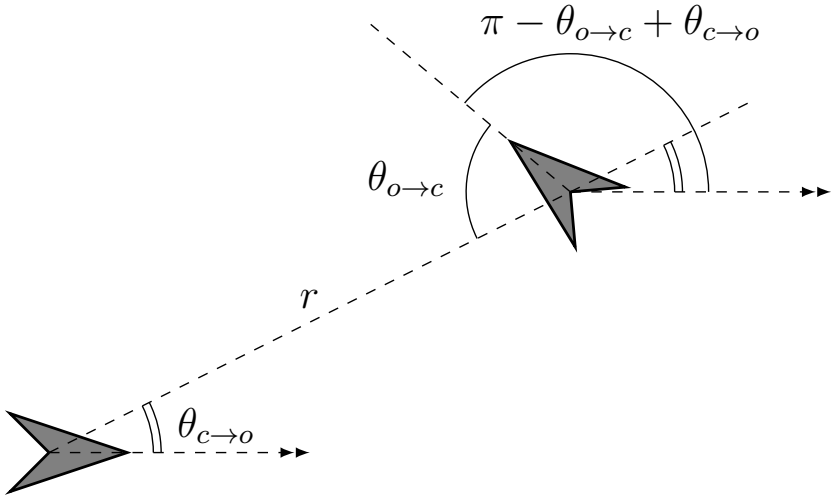


Figure 4.1: Diagram illustrating how the relative equations are determined.

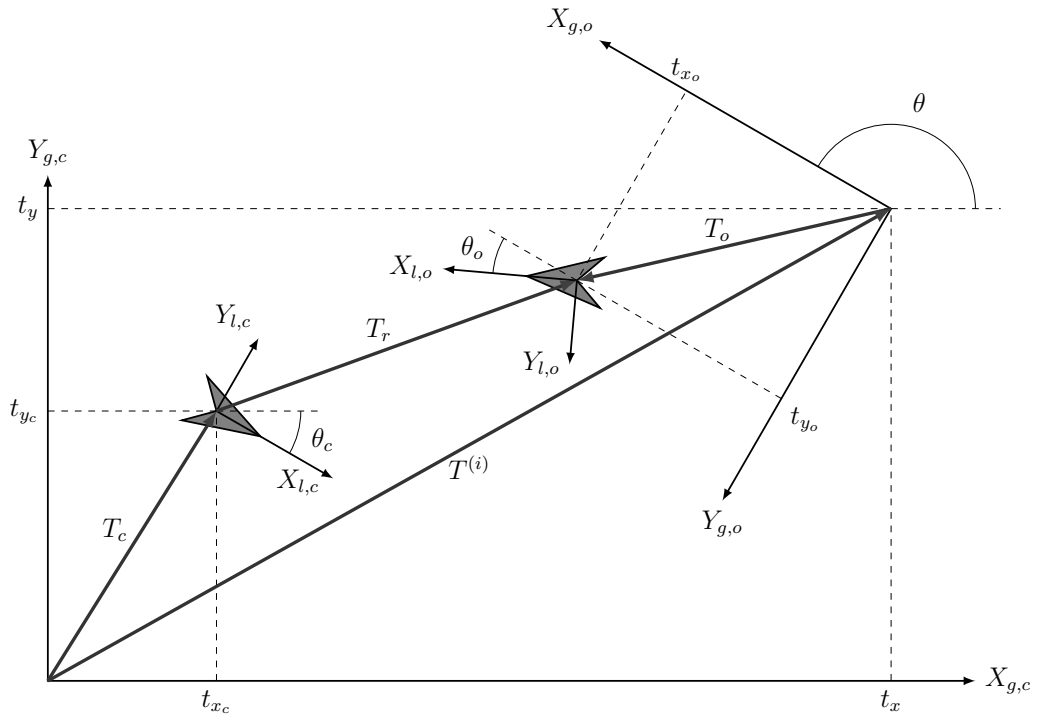


Figure 4.2: Diagram illustrating how transformation parameters and matrices have been determined.

each map the current robot's particles. This will obtain a better estimate of $T_c T_r$, which represents the transformation from the other robot's local frame of reference to the current robot's global frame of reference.

4.1.1 Improving Transformation Matrix Estimate

For feature maps, the $T_c T_r$ estimate can be achieved using a least squares estimate (LSE) based matching algorithm to match the other robot's recently perceived features to the current robot's individual particle map of features. This algorithm is based on the work of [20] and straightforward because it has already been assumed that the feature extractor is able to identify each feature. This simplication would be difficult to achieve in experiments due to the complexity of the data association problem [5]. Typically features are not easily identifiable and thus trying to associate the same feature in both the other robot most recent observation and the current robot's map would be compuationally difficult. Since this is outside the scope of this thesis, the data association problem has been assumed to be solved.

The following equation shows the error that needs to be minimized between the mutually seen features in the current robot's map and the other robot's recently observed features:

$$E = \sum_{j=1}^{M} [(x_c^j - \hat{x}^j)^2 + (y_c^j - \hat{y}^j)^2] \tag{4.3}$$

where x_c^j and y_c^j are the coordinates of a feature in the current robot's map, $\hat{x}^j$ and $\hat{y}^j$ are the estimated transformed coordinates from the other robot's observation, and M is the number of mutually observed features. Figure 4.3 shows the error values trying to be minimized from features that are mutually in the other robot's most recent observation and the current robot's map.

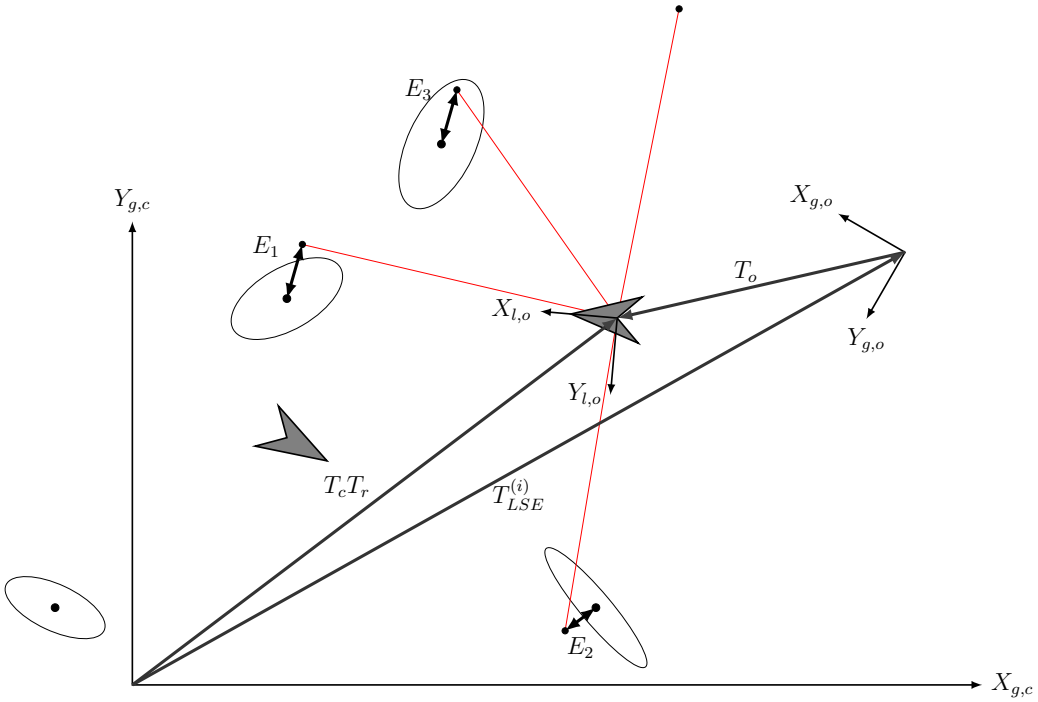


Figure 4.3: Diagram illustrating how the LSE can be used to improve the transformation matrix.

The estimated transformed coordinates are calculated as follows

$$\begin{bmatrix} \hat{x}^j \\ \hat{y}^j \end{bmatrix} = \begin{bmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{bmatrix} \begin{bmatrix} x_o^j \\ y_o^j \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{4.4}$$

where x_o^j and y_o^j are the coordinates of a feature in the observation of the other robot, and Δx, Δy, and $\Delta\theta$ are the transformation parameters. Minimizing the function in

Equation (4.3), yields the equation for the parameters

$$\begin{aligned}
&\frac{\partial E}{\partial \Delta\theta} = 0 \Rightarrow \\
&\Delta\theta' = \tan^{-1}\left(\frac{S_{x_c}S_{y_o} + MS_{y_c x_o} - MS_{x_c y_o} - S_{x_o}S_{y_c}}{MS_{x_c x_o} + MS_{y_c y_o} - S_{x_c x_o} - S_{y_c y_o}}\right) \\
&\frac{\partial E}{\partial \Delta x} = 0 \Rightarrow \\
&\Delta x' = \frac{S_{x_c} - \cos(\Delta\theta')S_{x_o} + \sin(\Delta\theta')S_{y_o}}{M} \\
&\frac{\partial E}{\partial \Delta y} = 0 \Rightarrow \\
&\Delta y' = \frac{S_{y_c} - \sin(\Delta\theta')S_{x_o} - \cos(\Delta\theta')S_{y_o}}{M}
\end{aligned} \tag{4.5}$$

where the S terms stand for the following sums

$$\begin{aligned}
&S_{x_c} = \sum_{j=1}^{M}[x_c^j], S_{x_o} = \sum_{j=1}^{M}[x_o^j], \\
&S_{y_c} = \sum_{j=1}^{M}[y_c^j], S_{y_o} = \sum_{j=1}^{M}[y_o^j], \\
&S_{x_c x_o} = \sum_{j=1}^{M}[x_c^j x_o^j], S_{x_c y_o} = \sum_{j=1}^{M}[x_c^j y_o^j], \\
&S_{y_c x_o} = \sum_{j=1}^{M}[y_c^j x_o^j], S_{y_c y_o} = \sum_{j=1}^{M}[y_c^j y_o^j]
\end{aligned} \tag{4.6}$$

This LSE algorithm is performed for each of the current robot's particles and leads to the following equation for the transformation matrix:

$$T^{(i)} = \begin{bmatrix} \mathrm{c}\,\Delta\theta' & -\mathrm{s}\,\Delta\theta' & \Delta x' \\ \mathrm{s}\,\Delta\theta' & \mathrm{c}\,\Delta\theta' & \Delta y' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{c}\,\theta_o & -\mathrm{s}\,\theta_o & t_{x_o} \\ \mathrm{s}\,\theta_o & \mathrm{c}\,\theta_o & t_{y_o} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \tag{4.7}$$

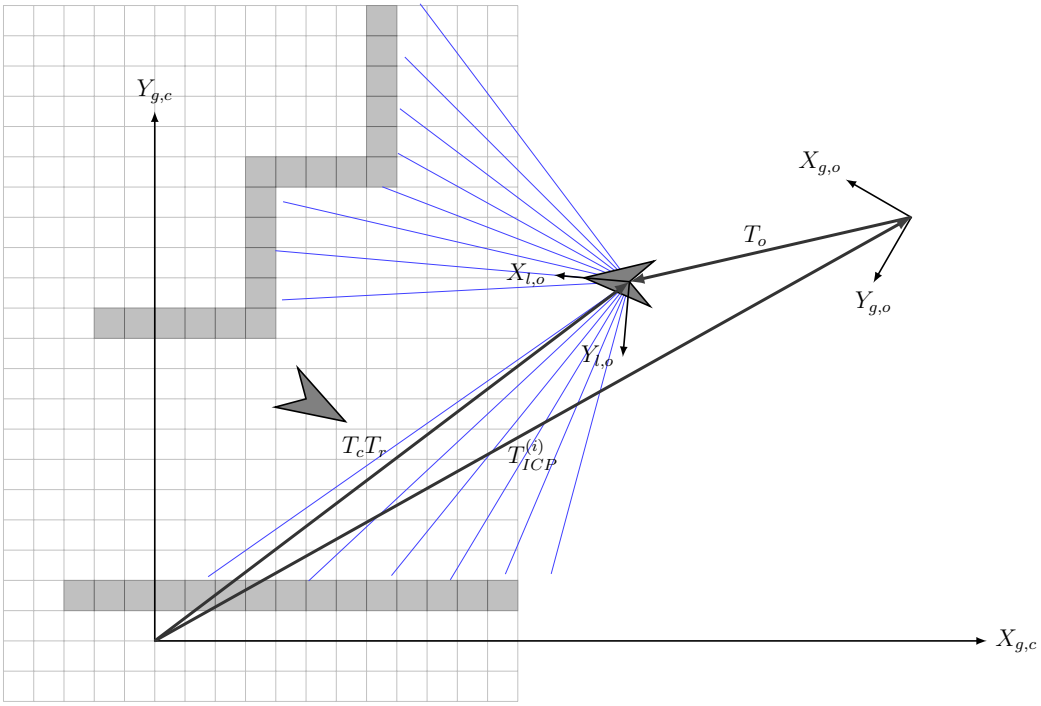


Figure 4.4: Diagram illustrating how the ICP can be used to improve the transformation matrix.

A similar algorithm for occupancy grid map named iterative closest point (ICP) scan matching [20] can be used to find a better approximation that attempts to fit the recent LRF observation from the other robot into each of the maps of the current robot's particles. This is shown graphically in Figure 4.4. Initially, ICP was designed to use a gradient descent method to revise the transformation, translation and rotation, parameters to minimize the distance, or error local minimum, between the points of raw LRF scans. In MRPT, it has been implemented to use KD-trees to accelerate the search and capable of matching a scan measurement to an occupancy grid map. KD-trees is short for k-dimensional trees and is a data structure for organizing points in a k-dimensional space [21]. This algorithm will yield a relative pose $(x_{ICP}, y_{ICP}, \theta_{ICP})$ that can then be inputted into the following equation for the

transformation matrix:

$$T^{(i)} = \begin{bmatrix} \mathrm{c}\,\theta_{ICP} & -\mathrm{s}\,\theta_{ICP} & x_{ICP} \\ \mathrm{s}\,\theta_{ICP} & \mathrm{c}\,\theta_{ICP} & y_{ICP} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{c}\,\theta_o & -\mathrm{s}\,\theta_o & t_{x_o} \\ \mathrm{s}\,\theta_o & \mathrm{c}\,\theta_o & t_{y_o} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \tag{4.8}$$

In some circumstances, it is possible for neither of these algorithms to reach an improved transformation. When this does occur, the initial estimate transformation matrix shown in (4.1) is used.

The merging process is summarized at the end of Algorithm 4.1 and depicted in Figure 4.5. Figure 4.5 shows how the current robot's maps are 'extracted' from the particles and then used in conjunction with the other robot's observation to get $T_c T_r$. This result can then be combined with T_o^{-1} to transform the other robot's map. The transformed map can then be merged with the current robot's maps and subsequently 'inserted' back into the current robot's particles.

4.2 Model Selection

Reinforcement learning is the method being used to determine when to best merge maps and which method to use to improve the transformation matrix. However, to successfully accomplish this, an appropriate model must first be selected that is simple enough and yet has access to enough data to converge to the proper decision matrix. This data will be based on the current status of the robot particle filters and the current overlap in robot sensor readings. The reward signal being properly chosen is also critical.

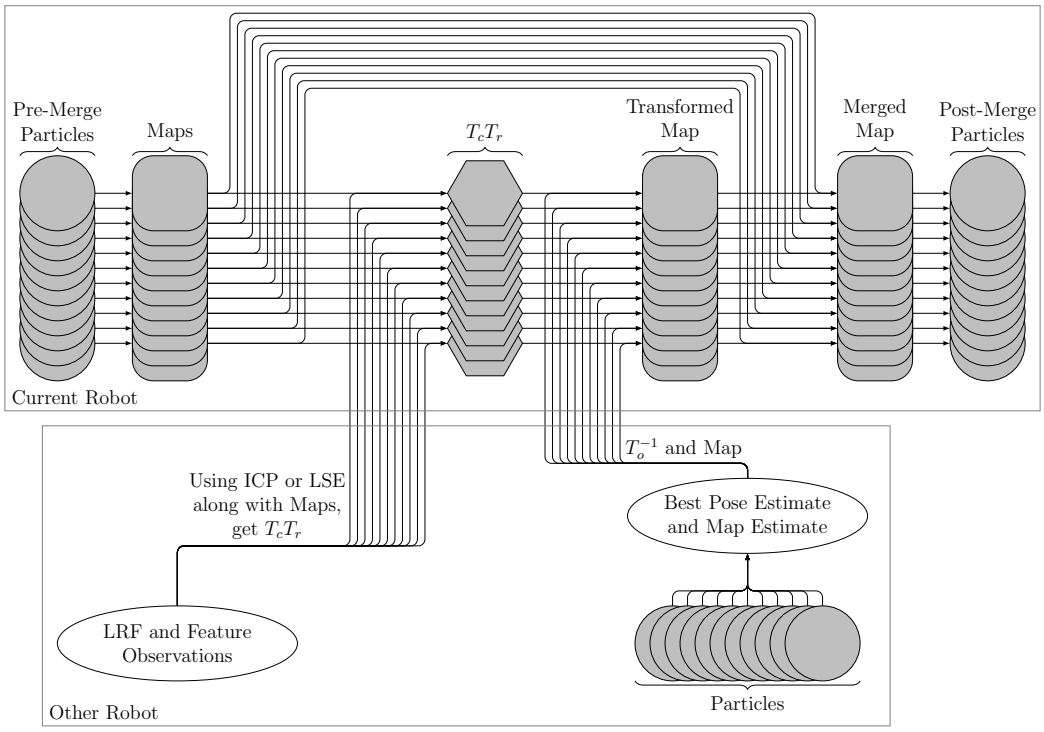


Figure 4.5: Diagram demonstrating the process of how the maps produced from FastSLAM by each individual robot are merged from one robot's perspective.

4.2.1 N_{eff} Criterion

Each particle i of a robot's particle filter also has an attributed weight $w^{(i)}$ which indicates the likeliness of the particle's pose and current maps. These weights cannot be directly compared time step to time step due to the fact that they are always normalized. However, there is an achievable measure of evaluation by calculating the weight distribution over the particle filter. This value can be a heuristic in the choice of whether or not a merge is recommended.

The weight distribution can be measured by the variance of the weights. For example, when the filter does not have a high confidence in its particles' estimates the variance of a set of weights will be low, and vice versa. It is apparent that in most cases, for a merge to incur the least amount of error, the robots should be confident in their current set of particles. Especially since in section 4.1 the other robot chooses

its pose and maps candidates based on the particle with the greatest weight.

The effective sample size value, N_{eff}, from (2.12) provides a good basis for a decision criterion and is repeated here:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w^{(i)})^2}. \tag{4.9}$$

As stated previously upon inspection, one can deduce that the higher the variance in the weights will yield a lower N_{eff} value, and vice versa. As previously mentioned, many particle filters use this value for their resampling condition in order to remove lower weighted particles when certain particles have a much higher relative weight or when $N_{eff} < N/2$. Therefore, it seems reasonable to have a heuristic that says to merge when both robots' particle filters are resampling concurrently or when $N^c_{eff} < N/2$ and $N^o_{eff} < N/2$.

4.2.2 Mutual Observation Likelihood Criterion

LSE or ICP are used to improve upon the estimate of $T_c T_r$, a good heuristic would be to use the value of the confidence of this improved estimate. This confidence value could be calculated as the likelihood of the other robot's sensor readings into the current robot's maps.

For both grid-based and feature-based methods, this mutual observation likelihood is calculated as the probability of the other robot's measurement reading, $z_{o,t}$, given the transformed pose based on the improved relative transformation matrix, $T_c T_r x^{\dagger}_{o,t}$ in each of the current robot's particles maps, $m^{(i)}_{c,t}$. This is summarized by

$$l\left(z_{o,t}, T_c T_r x^{\dagger}_{o,t}, m_{c,t}\right) = p\left(z_{o,t} | T_c T_r x^{\dagger}_{o,t}, m_{c,t}\right). \tag{4.10}$$

where l represents a likelihood and p represents a probability.

The calculation of these probabilities uses the same techniques as discussed in section 2.2.2 for the calculation of the particle weights. The likelihood of the other robot's feature observation given the transformed pose and feature map of the current robot is:

$$p\left(z_{o,t}|T_cT_rx_{o,t},m_{c,t}\right)=\frac{\sum_{j=1}^{F_{\text{mut}}}\exp\left(-\frac{1}{2}(z_{o,f,t}^{(j)}-\hat{z}_{f,t}^{(j)})Q_t^{-1}(z_{o,f,t}^{(j)}-\hat{z}_{f,t}^{(j)})\right)}{F_{\text{mut}}} \tag{4.11}$$

where F_{mut} is the number of features that are both in the other robot measurement and the current map $m_{c,t}^{(i)}$; $z_{o,f,t}^{(j)}$ is the other robot measurement of a feature; $\hat{z}_{o,f,t}^{(j)}=h(\mu_{c,j,t},T_cT_rx_{o,t}^{\dagger})$ is the measurement prediction; and $Q_t=H\Sigma_{c,j,t}H^T+Q$ is the measurement covariance with $H=h'(\mu_{j,t-1}^{(i)},x_t^{(i)})$ being the Jacobian of the measurement model and Q is the measurement sensor noise covariance. In this case the measurement prediction is calculated as:

$$h(\mu_{c,j,t},T_cT_rx_{o,t}^{\dagger})=\begin{bmatrix}\sqrt{(\mu_{c,j,t,x}-x_{T_cT_r})^2+(\mu_{c,j,t,y}-y_{T_cT_r})^2}\\ \operatorname{atan2}\left(\mu_{c,j,t,y}-y_{T_cT_r},\mu_{c,j,t,x}-x_{T_cT_r}\right)-\theta_{T_cT_r}\end{bmatrix} \tag{4.12}$$

where $(x_{T_cT_r},y_{T_cT_r},\theta_{T_cT_r})^T=T_cT_rx_{o,t}^{\dagger}$. The calculation of H is similar to (2.8) and will not be repeated here.

Determining the confidence of the improved transformation from the grid-based approach can be done using the 'beam endpoint model' [5] whose method is outlined in Algorithm 2.6. As previously explained, the ray endpoints from the other robot's laser data would be transformed and place into a likelihood field of the current robot's occupancy grid map. The product of these ray likelihood will produce the requested grid-based mutual observation likelihood.

The previous equations neglected the fact that these calculations must be done on all the particles. Therefore, all these values must be averaged in order to reach a final value that can be used in the heuristic to decide which method should be used to improve the transformation matrix and if a merge is recommended:

$$\bar{l} = \frac{1}{N} \sum_{i} l \left(z_{o,t}, T_c T_r x_{o,t}^{\dagger}, m_{c,t}^{(i)} \right) \tag{4.13}$$

Independently, these heuristic values can be used to determine when to merge if the value is above a certain threshold ($\bar{l} \geq c$). This threshold value must be sufficiently high as not to incur much error when merging maps. If there is a case where both feature- and grid-based threshold values are above their respective constants, then the greater of the two should be used even though they are not directly comparable. This is an obvious pitfall of this particular heuristic, but it should be noted that this is simply a heuristic and subsequent discussions will show how this particular pitfall is avoided using Reinforcement Learning.

4.2.3 Reinforcement Learning for Model Selection

Using a similar method used in [12] described in section 2.3, the previous two heuristics can be combined using Reinforcement Learning to take advantage of their strengths while avoiding their pitfalls. The requirements include the proper definition of the states S, the actions A, and the reward $r : S \rightarrow R$. The straightforward actions are $A = a_{dm}, a_{mg}, a_{mf}, a_{mi}$, where a_{dm} defines the action of not merging, a_{mg} defines the action of merging using the grid-based method to improve the transformation matrix, a_{mf} defines the action of merging using the feature-based method to improve the tranformation matrix, and a_{mi} defines the action of merging using the initial estimate of the transformation matrix to merge the maps.

Determine Proper State Composition

As previously mentioned, the state set must be defined to represent enough of the information from the sensors and particle filters, e.g. the environment, to make an 'educated' decision. The first reinforcement learning state variable is a boolean representing the N^c_{eff} criterion for the current robot, i.e. $N^c_{eff} < N/2$. Similarly, the second state variable is a boolean representing the N^o_{eff} criterion for the other robot, i.e. $N^o_{eff} < N/2$. The next state variable is the confidence of the ICP recommended transformation matrix, given by the grid-based mutual observation likelihood $\bar{l}_g$. Finally, the last state variable is the confidence of the LSE recommended transformation matrix, given by the feature-based mutual observation likelihood $\bar{l}_f$. The values of $\bar{l}_g$ and $\bar{l}_f$ are divided into three discrete intervals (0.0-0.3333, 0.3333-0.6666, 0.6666-1.0). Therefore, the resulting state set is

$$S := \bar{l}_g \times \bar{l}_f \times 1_{N^c_{eff}<N/2} \times 1_{N^o_{eff}<N/2}. \tag{4.14}$$

From (4.14), the number of states is $3 \times 3 \times 2 \times 2 = 36$ states. The larger the state set is, the lengthier the necessary training will be, thus it is necessary to keep the number of states as low as possible while not affecting the accuracy of the training.

It should be noted that there exists other possible states in this model, which include the state of: pre-rendezvous, post-rendezvous, and post-merge. However, these can be eliminated since there is only one possible action, that is of not merging, and can be set to have the reward signal of zero. Figure 4.6 depicts a state diagram showing all of the states and the possible transitions between them.

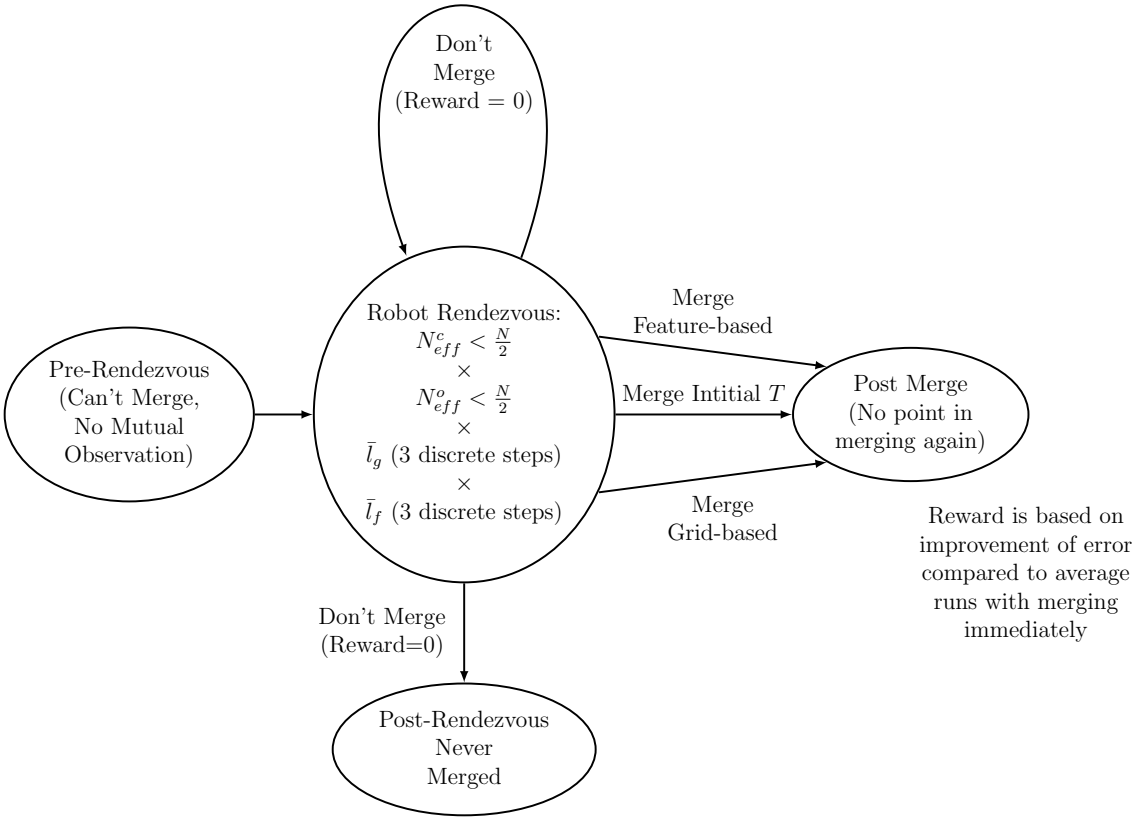


Figure 4.6: State Diagram demonstrating how the Reinforcement Learning states transition between one another.

Choosing Appropriate Reward Signal

Without the proper selection of a reward signal the training using the SARSA algorithm will never converge to a decision matrix or worse will converge to an undesirable decision matrix. Since the training is being learned from simulated data, the true robot pose x_t^* is known at every time step t and should be included in any attempted reward signal proposition. The reward signal that yielded the best convergence results was based on the amount of improvement (or deviation which yields a negative reward) in the normalized cumulative error for the entire simulated run and is calculated by:

$$r(s_t) = E_{cum}^{\mu} - E_{cum} \tag{4.15}$$

where $r(s_t)$ is the reward for the current state s_t, E_{cum}^{μ} is the average cumulative error achieved by several runs where the robots immediately merge upon mutual observance. The normalized cumulative error over an entire run is calculated by the following equation:

$$E_{cum} = \sum_{i=1}^{t} ||x_i - x_i^*|| = \sum_{i=1}^{t} \sqrt{\Delta x^2 + \Delta y^2 + \Delta\theta^2} \tag{4.16}$$

where x_i with $i = 1 \rightarrow t$ is the entire pose trajectory from the particle with greatest weight at the end of the simulated run, and (Δx, Δy, $\Delta\theta$) are the individual pose components. Although not all components have the same units, their values were typically within an order of magnitude and thus could be normalized together without one of the components providing too much bias.

The choice of focussing on trajectory error is based on the fact that a best possible map of the environment can be achieved when the estimated robot trajectory has zero error. Therefore, any reduction should be rewarded.

Training

During training, the SARSA Reinforcement Learning algorithm, described in Algorithm 2.9, will cause the Q values for each state-action to converge and produce a decision matrix. This decision matrix will allow the current robot to only merge its maps with the other robot's at a state that should reduce the eventual trajectory error.

The choice of training environment is critical to determine a proper decision matrix. Similar to the work of Wurm *et al.* in [1], the environment consists of an indoor section with a building-like structure containing rooms and hallways and an outdoor section with features that model a set of trees or posts. In order to ensure each training episode is not too long, only one merge is allowed per episode and multiple different robot rendezvous episodes would be used to ensure that all reinforcement learning states were visited. Figure 4.7 shows the four different scenarios that were used in the training episodes with the gray lines representing the desired robot trajectories. In one scenario the robots are always travelling within the simulated building; in another both robots are always in the simulated area of trees; in another one robot starts outside while the other robot starts inside and they meet in the entrance of the building; and similarly one robot starts outside and the other inside and they rendezvous near an outside corner of the building. Since this thesis is based on a decentralized algorithm, each robot's perspective could be used in a training episode.

In order to speed up the training episodes, the pre-rendezvous sections of the training were processed and the particle filter status is stored. This way the training could take place immediately from the time step where merging was possible. However, this caused certain elusive states to not be visited. Therefore, ten setups of each scenario were stored to increase the chance of visiting each reinforcement learning state, bringing the total number of variants to $4 \times 2 \times 10 = 80$. Thousands of

episodes were performed using these available setups until the decision matrix finally converged satisfactorily.

During training, an ε-greedy policy was used, where in a particular state s, the policy would choose action a which had the highest value $Q(s, a)$, but another random action would be explored with probability ε. To allow suitable exploration of the possible state-action pairs, a value of $\varepsilon = 0.1$ was used during training. The learning rate α was set to 0.001 and the discounting factor γ was set to 0.9, which according to [1] are standard values. They also lead to good results in this training.

In terms of particle filter parameters, the policy determined through validating [1] was used to determine which proposal should be used in propagating particles and the number of particles was set to $N = 25$. The rest of the parameters used by the particle filter and map types are shown in Appendix C as the 'ini' config file used by the MRPT dataset processing software.

The overall mapping and map-merging algorithm that is to be used once training is complete is shown in Algorithm 4.1.

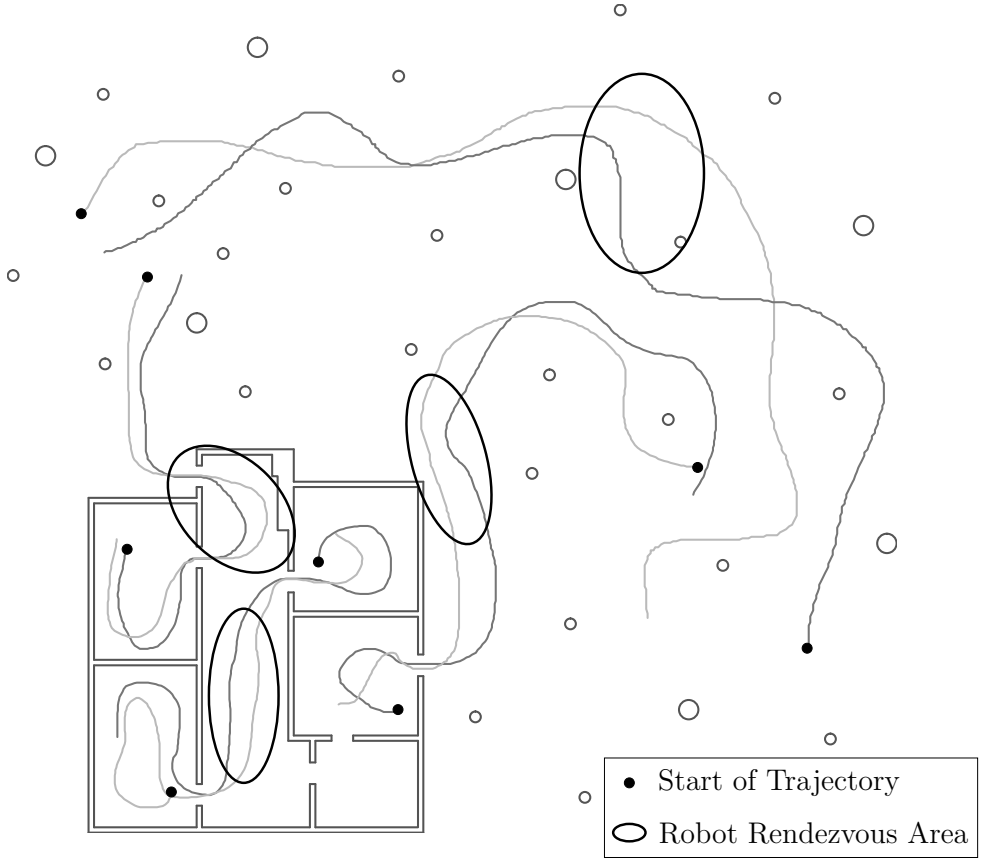


Figure 4.7: Training Environment with building and set of trees with dimensions 45 [m] ×45 [m]

Algorithm 4.1 Mapping and Map-Merging approach

Require:
S_{t-1}, previous time step sample set of particles for current robot
$z_{f,t}$, most recent feature observations for current robot
$z_{g,t}$, most recent laser scan for current robot
u_{t-1}, most recent odometry measurement for current robot
$z^o_{f,t}$, most recent feature observations for other robot
$z^o_{g,t}$, most recent laser scan for other robot
N^o_{eff}, most recent weight variance from other robot
Ensure:
S_t, new sample set

maptype = decideProposal$(S_{t-1}, z_f, z_g, u_{t-1})$

$S_t = \{\}$ *// Start with Empty Set*
for all $s^{(i)}_{t-1} \in S_{t-1}$ **do**
$\langle x^{(i)}_{t-1}, w^{(i)}_{g,t-1}, w^{(i)}_{f,t-1}, m^{(i)}_{g,t-1}, m^{(i)}_{f,t-1} \rangle = s^{(i)}_{t-1}$

if maptype=grid **then** *// Compute proposal*
$x^{(i)}_t \sim p(x_t | x^{(i)}_{t-1}, u_{t-1}, z_{g,t})$
else
$x^{(i)}_t \sim p(x_t | x^{(i)}_{t-1}, u_{t-1}, z_{f,t})$
end if

$w^{(i)}_{g,t} = \text{updateWeight}(w^{(i)}_{g,t-1}, m^{(i)}_{g,t-1}, z^{(i)}_{g,t})$
$w^{(i)}_{f,t} = \text{updateWeight}(w^{(i)}_{f,t-1}, m^{(i)}_{f,t-1}, z^{(i)}_{f,t})$
$m^{(i)}_{g,t} = \text{integrateScan}(m^{(i)}_{g,t-1}, x^{(i)}_t, z^{(i)}_{g,t})$
$m^{(i)}_{f,t} = \text{integrateFeatures}(m^{(i)}_{g,t-1}, x^{(i)}_t, z^{(i)}_{f,t})$

// update sample set
$S_t = S_t \cup \{\langle x^{(i)}_t, w^{(i)}_{g,t}, w^{(i)}_{f,t}, m^{(i)}_{g,t}, m^{(i)}_{f,t} \rangle\}$
end for

for $i = 1$ to N **do**
if maptype=grid **then**
$w^{(i)} = w^{(i)}_{g,t}$
else
$w^{(i)} = w^{(i)}_{f,t}$
end if
end for

$\vdots$

Algorithm 4.1 *Continued*

$N_{eff} = \frac{1}{\sum_{i=1}^{N}(w^{(i)})^2}$
if $N_{eff} < N/2$ **then**
  $S_t = \text{resample}(S_t, \{w^{(i)}\})$
end if

mergetype = decideMerge($N_{eff}, N^o_{eff}, z^o_{f,t}, z^o_{g,t}, m_{f,t}, m_{g,t}$)

if mergetype = mergeGrid || mergeFeature **then**
  $\langle m^o_{f,t}, m^o_{g,t}, x^o_t \rangle$ = otherMostLikMapAndPose()
  for $i = 1$ to N **do**
    if mergetype=mergeGrid **then**
      $T^{(i)}$ = getTransMatrixUsingICP($z^o_{g,t}, m^{(i)}_{g,t}, x^o_t$)
    else
      $T^{(i)}$ = getTransMatrixUsingLSE($z^o_{f,t}, m^{(i)}_{f,t}, x^o_t$)
    end if
    $m^{(i)}_{g,t}$ = mergeGridMaps($m^{(i)}_{g,t}, T^{(i)} m^o_{g,t}$)
    $m^{(i)}_{f,t}$ = mergeFeatureMaps($m^{(i)}_{f,t}, T^{(i)} m^o_{f,t}$)
  end for
end if

Chapter 5

Simulations

Section 4.2.3 outlines the final parameters used in our proposed approach, these parameters came about from many iterative trials. The computational requirement of running a training trial were high, on the order of several continuous days of computation on a Quad Core i7 processor, and thus limited the number of trials attempted for this thesis. The values varied during different experiments were modified action sets, ε-greedy values, and reward signals.

The original action set did not include the action of merging with the initial transformation matrix, a_{mi}. However it was deemed prudent to add this action in order to properly evaluate the proposed improved transformation matrix estimates.

For our simulations, a value of $\varepsilon = 0.1$ provided the necessary exploration of reinforcement learning states.

The choice of suitable reward signal required the most experimentation. The first reward signal attempted compared the total normalized cumulative error (TNCE) for a certain episode to the lowest TNCE achieved so far. This choice led all state-action pairs to becomes negative and never allowing the Q values to converge to a suitable decision matrix. The next reward signal attempted involved comparing the current episode TNCE to the average TNCE of several runs where the robots never merged

their maps. Although this seemed to lead to Q values converging to desirable values, it was found that validation of these results did not prove successful. Finally, the reward signal based on the current episode TNCE compared to the average TNCE of several runs where the robots merged their maps immediately upon mutual observance. This choice is in line with the premise of this thesis, i.e. decreasing the amount of error incurred by merging, and yielded the best results.

5.1 Results

Once the Q values reached a suitable convergence, the training program was ended. The resulting trained policy is shown in Figure 5.1 as a decision tree created using the ID3 algorithm [22]. Logically, the most popular option is to not merge and to wait until a suitable state is reached where merging is the preferred option. The most popular merging option was that of the feature-based method. The grid-based and initial transformation matrix estimate method were only found to be recommended in one state each. It can be noticed in Figure 5.1 that generally for a merge to be recommended only one of the two robots have their particle filters with the $N_{eff} < N/2$. This is beneficial since it was also observed that the occurrence of both robots' particle having such a value was not common.

The subsequent values were then tested on the same training data. Figure 5.2 and Figure 5.3 show some sample results produced by using the training policy with the training data used to derive the policy. These results are calculated using the average of several runs through the same data. It can be seen that once the merge occurs, shown by the separation of the two curves, that our method performs better than merging immediately. Any amount of improvement is considered successful due

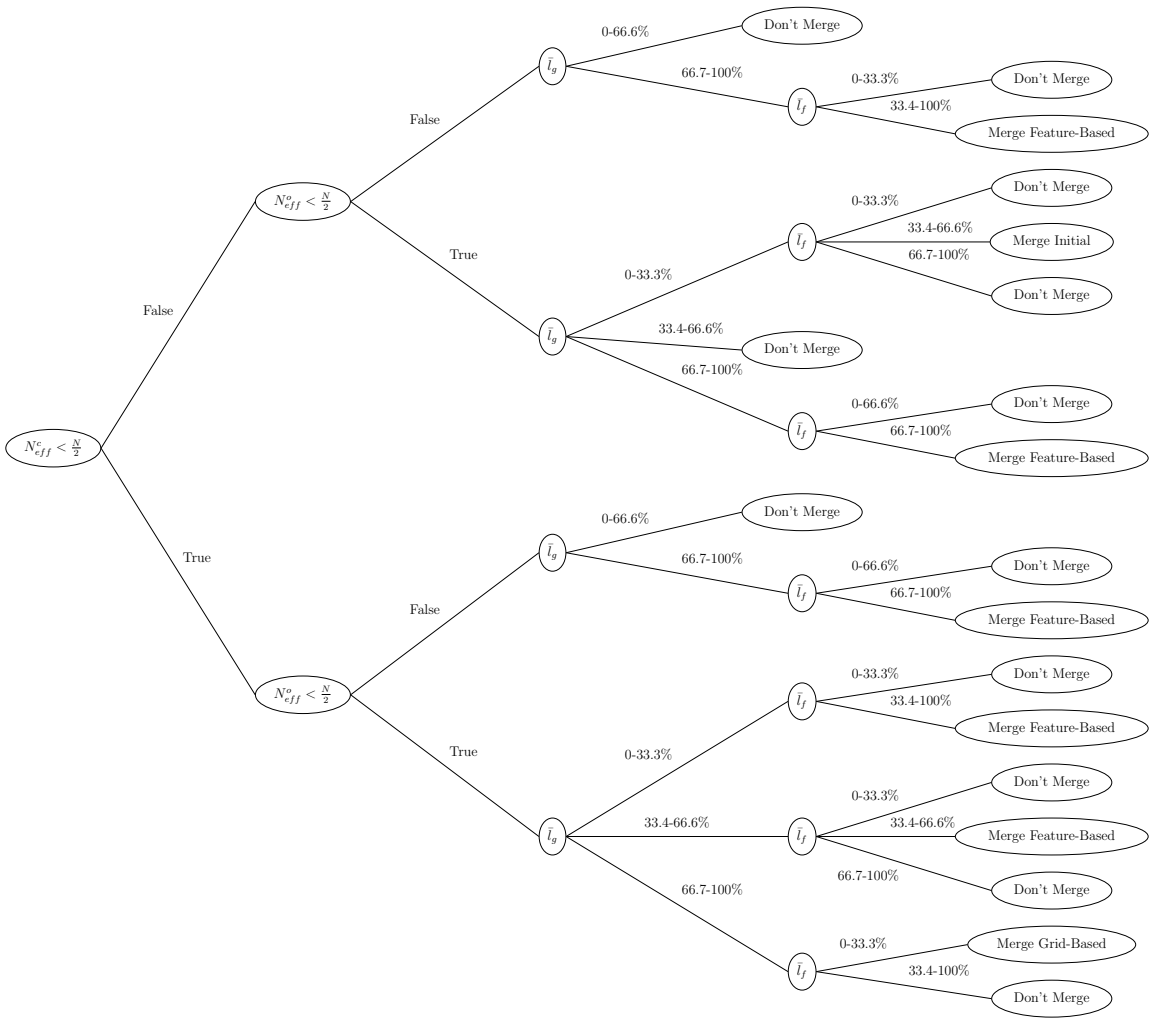


Figure 5.1: Decision Tree showing the policy determined through reinforcement learning

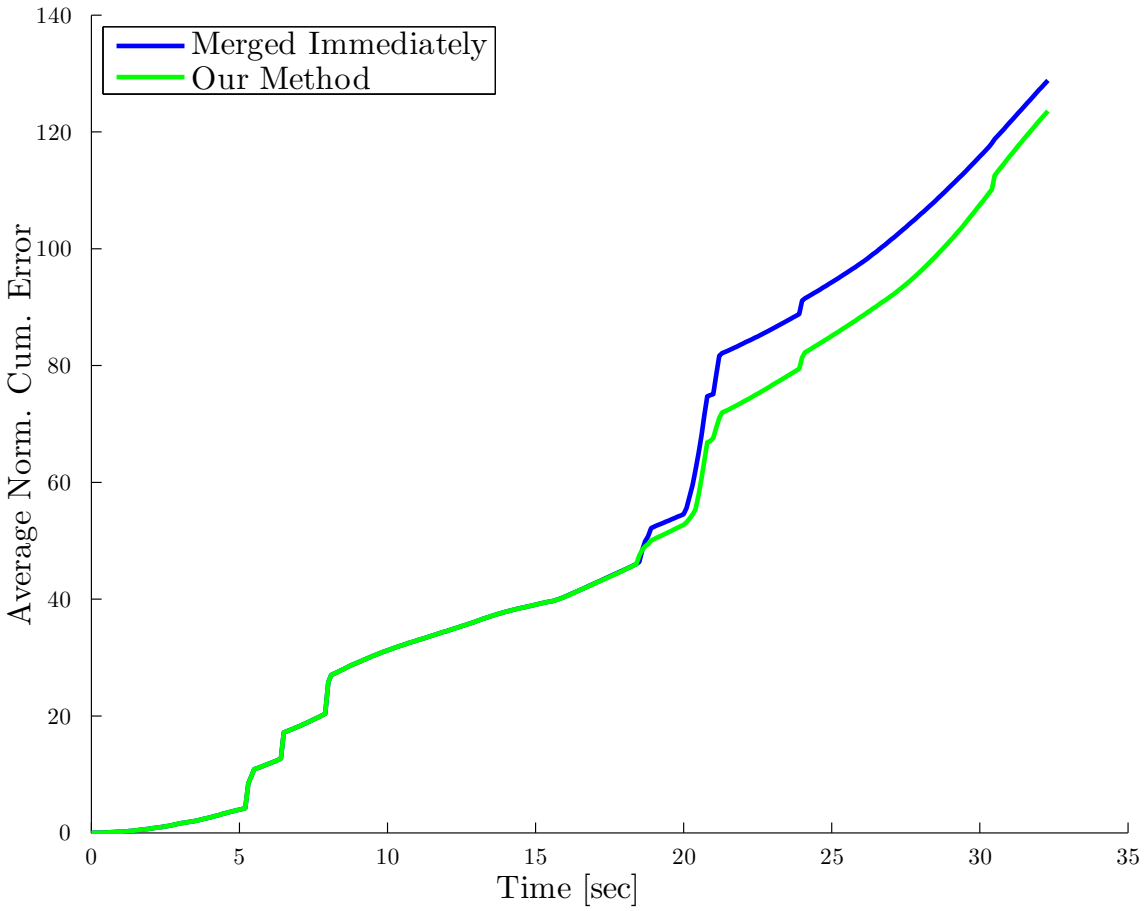


Figure 5.2: Comparison of cumulative error for one entire training data robot trajectory

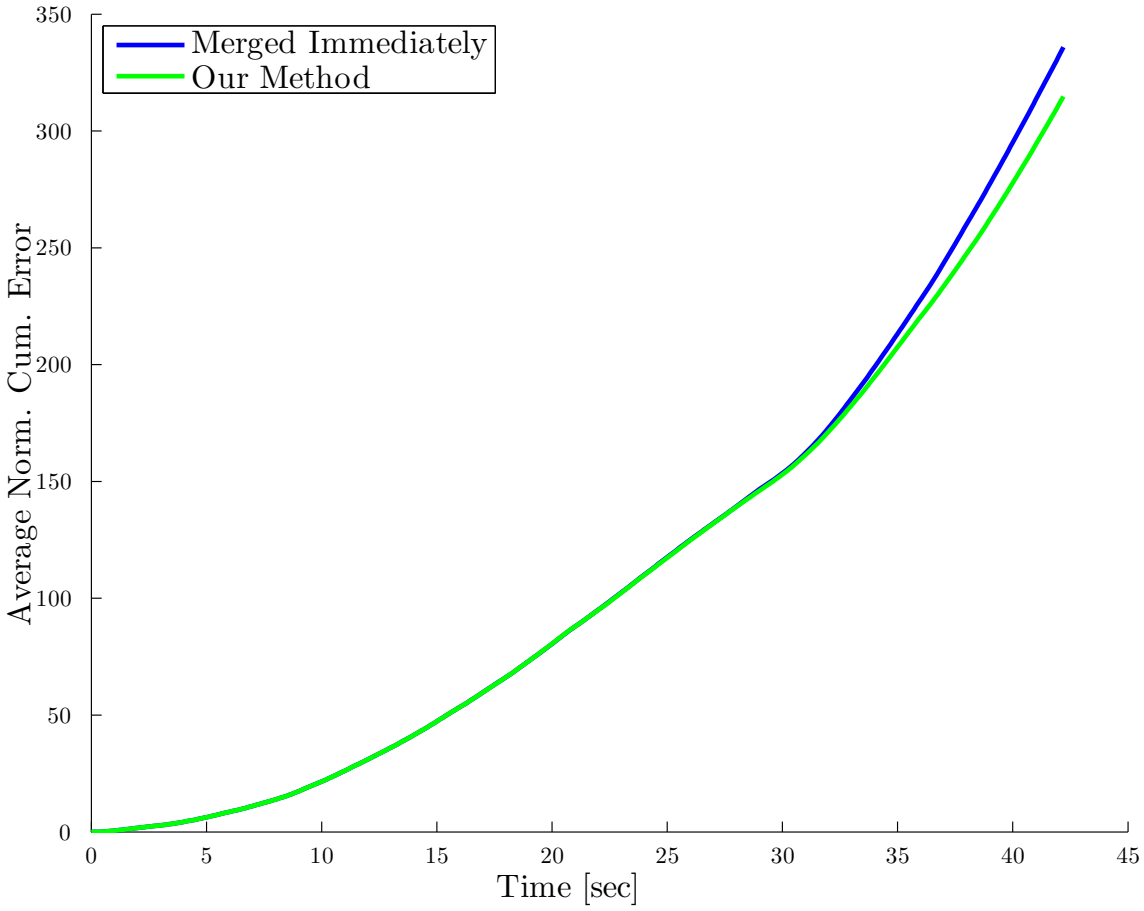


Figure 5.3: Comparison of cumulative error for another entire training data robot trajectory

to the desire to get as close as possible to zero error.

5.2 Validation

In order to properly validate the training results, a new environment was designed and new robot datasets were generated. Figure 5.4 shows the different environment along with some robot trajectories used for validation. This environment contains many of the same characteristics as the training environment, such as a building with

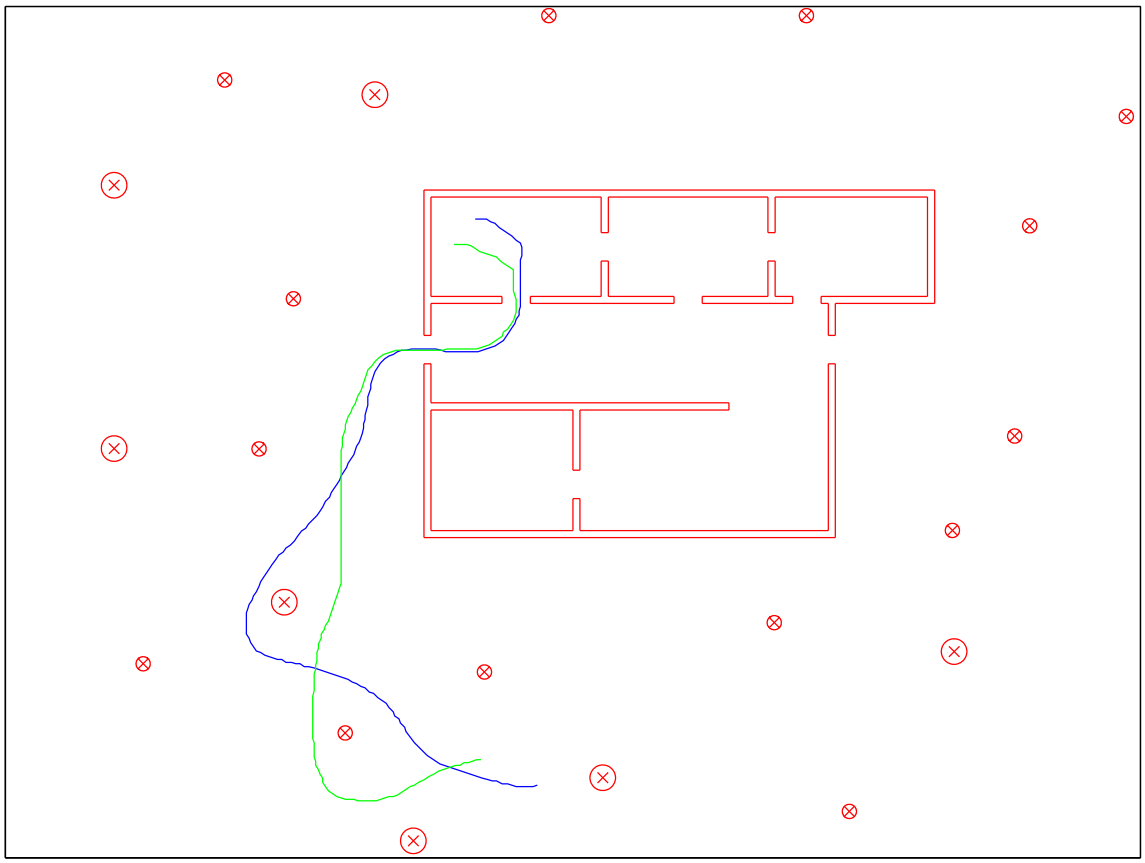


Figure 5.4: Environment used for validation with dimensions 45 [m] ×55 [m]

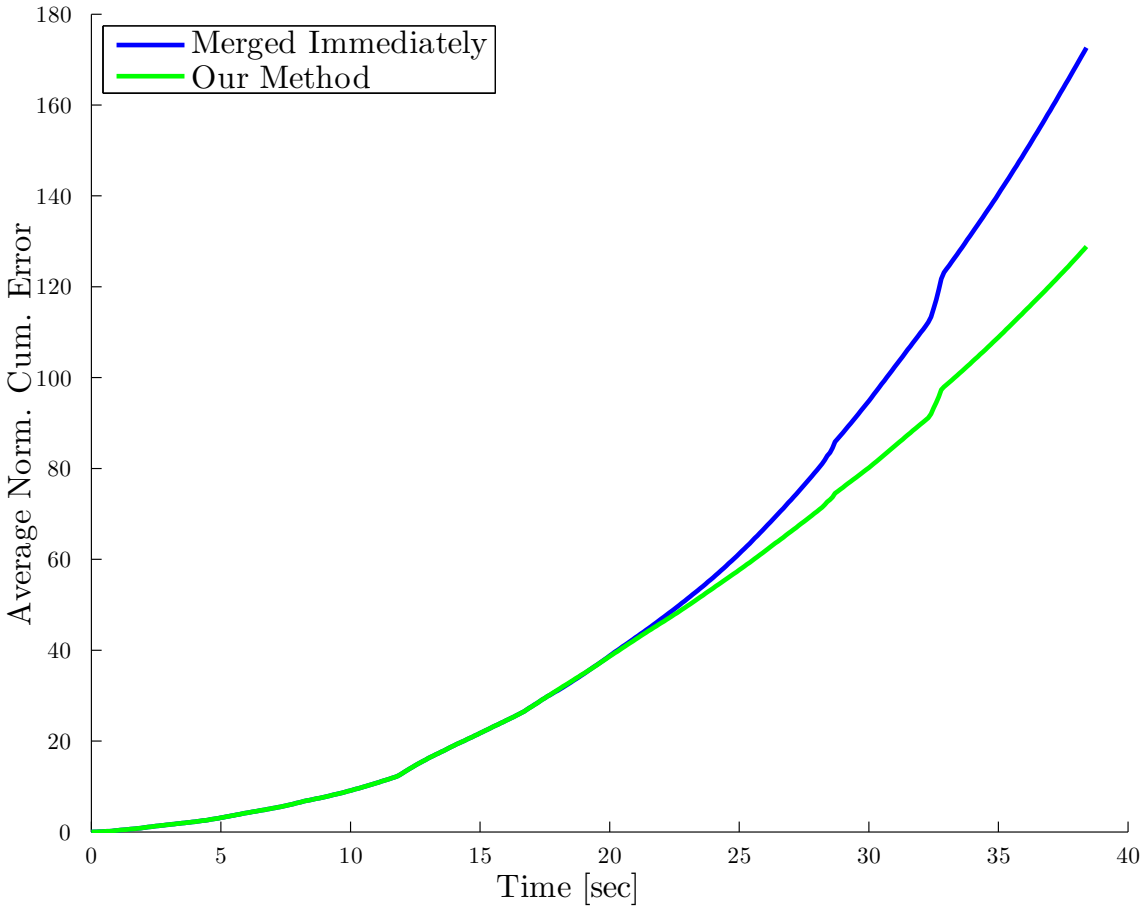


Figure 5.5: Comparison of cumulative error for one entire validation data robot trajectory

rooms and a large area covered with 'trees'.

Figure 5.5 and Figure 5.6 show some successful sample results achieved through validation. It can easily be seen that once the merge occurs, shown by the separation of the two curves in the figures at $t = 17.6$ seconds, that on average our method lowers the amount of incurred error.

Some sample final maps produced by the same validation scenario are shown in Figure 5.7 and Figure 5.8.

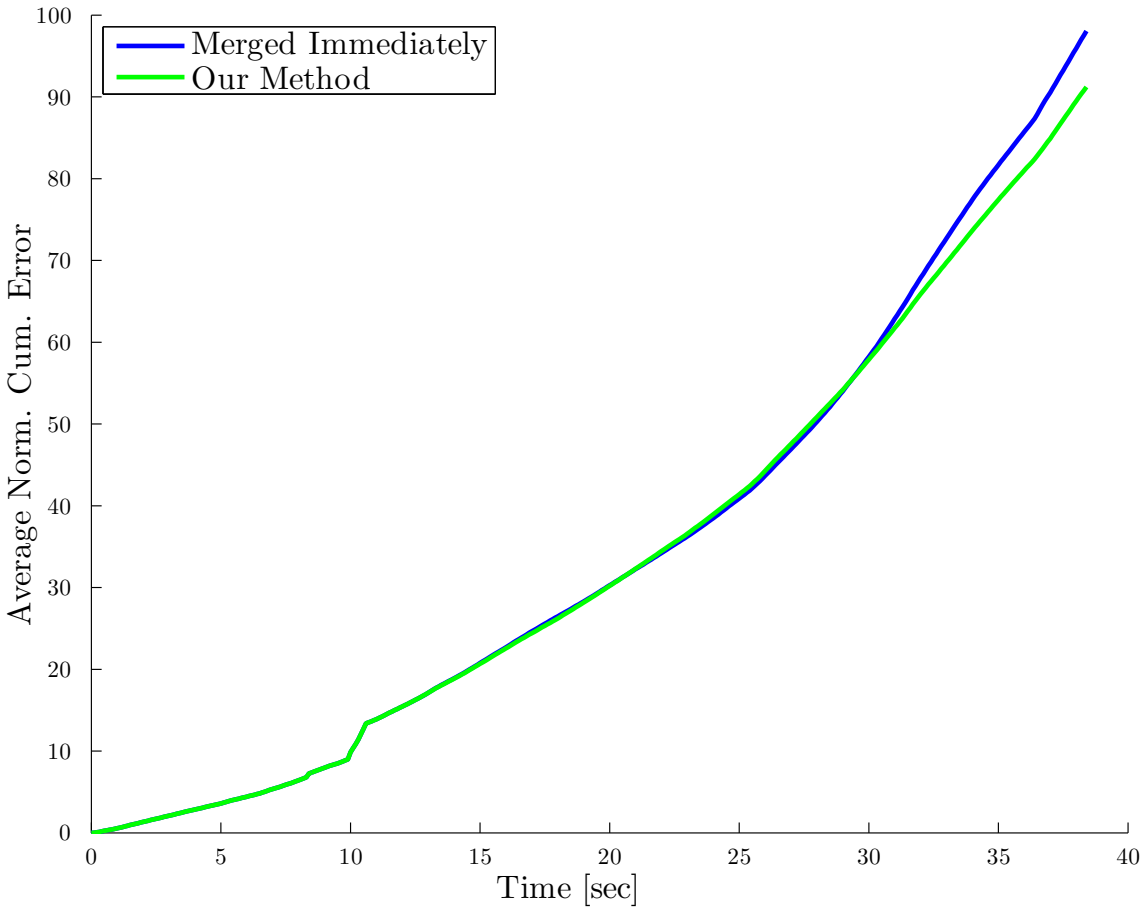
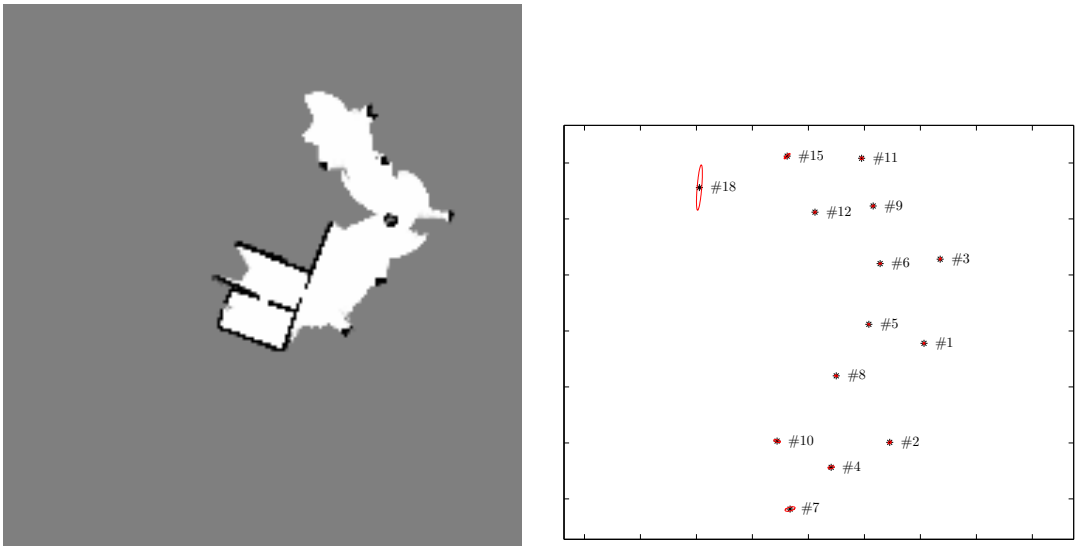
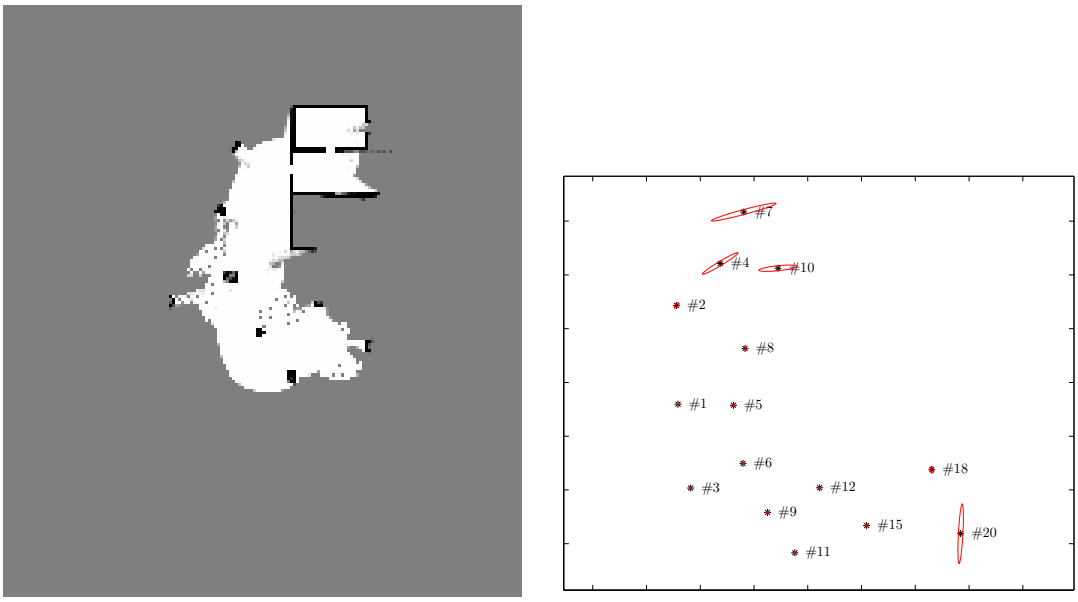


Figure 5.6: Comparison of cumulative error for one entire validation data robot trajectory



(a) Occupancy Grid Map (b) Feature Map

Figure 5.7: Sample maps produced during validation



(a) Occupancy Grid Map (b) Feature Map

Figure 5.8: Sample maps produced during validation

5.3 Single-Robot vs Multi-Robot SLAM

To show the benefits of Multi-Robot SLAM, a mapping area was selected that would be mapped by a single robot and then by multiple robots. The environment used during validation was also used for this purpose, this environment and the accompanying robot trajectories are shown in Figure 5.9. The single robot starts outside, eventually makes its way into the building, then upon exiting the building explores the outside. Conversely in the multi-robot run, 'Robot 1' starts inside and 'Robot 2' outside. The two robots meet just outside the building, and assuming that 'north' is vertically upwards on the map shown in Figure 5.4 'Robot 1' heads 'southwards' and 'Robot 2' 'northwards'.

Figure 5.10 shows the average cumulative error acculumated during several mapping runs. The rate of increase in cumulative error tends to increase as time increases, thus it makes sense that the final error value for the single robot is much greater than the final error values for the two robots.

Figure 5.11 and Figure 5.12 show the best final maps produced by both scenarios respectively. It can be seen in Figure 5.11(a) that the single robot had accumulated too much error by the time it entered the building and caused the mapping of the building to be filled with inconsistencies. An example of an inconsistency in an occupancy grid map is when a wall or corner appears multiple times in the same map. Conversely, in Figure 5.12(a) the mapping of the building appears to be more consistent with what was expected. The exponential growth in the average normalized cumulative error is typical in SLAM applications where there is no available sensor that can give a global reference frame reading, i.e. a GPS type sensor. Which is why

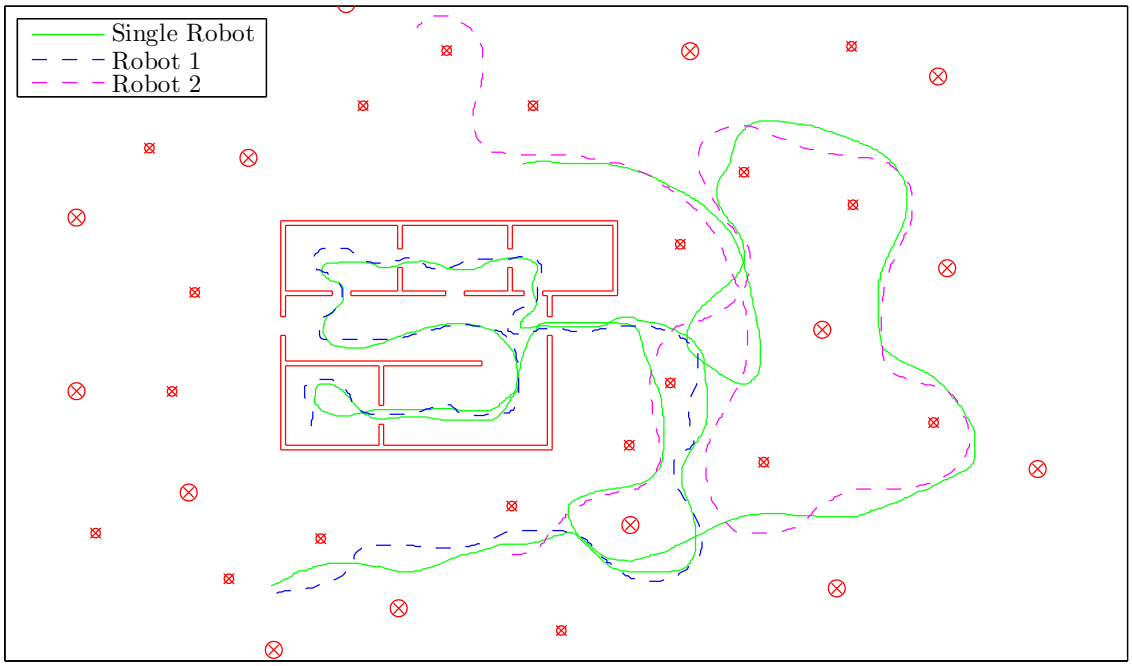


Figure 5.9: Trajectories used in comparing a single robot mapping compared to multi-robot mapping

there is ongoing research in attempting to reduce this curve as much as possible.

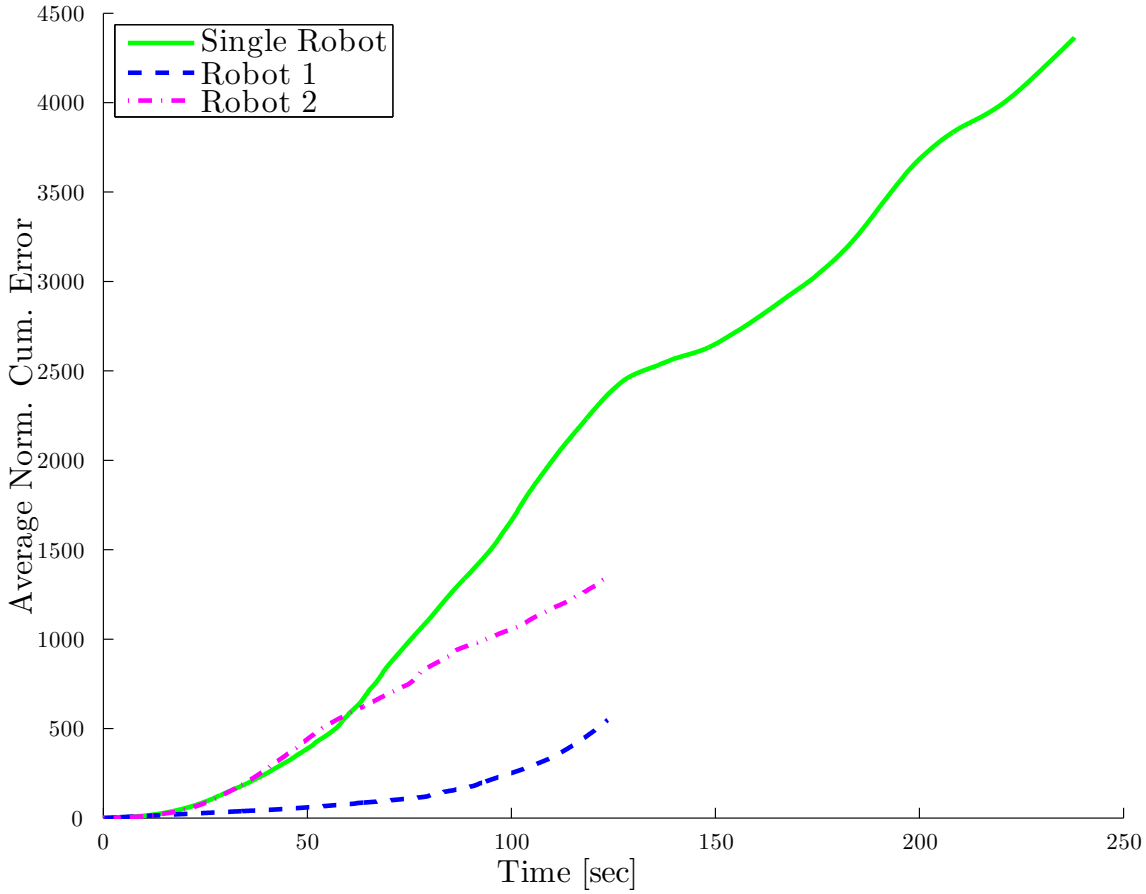
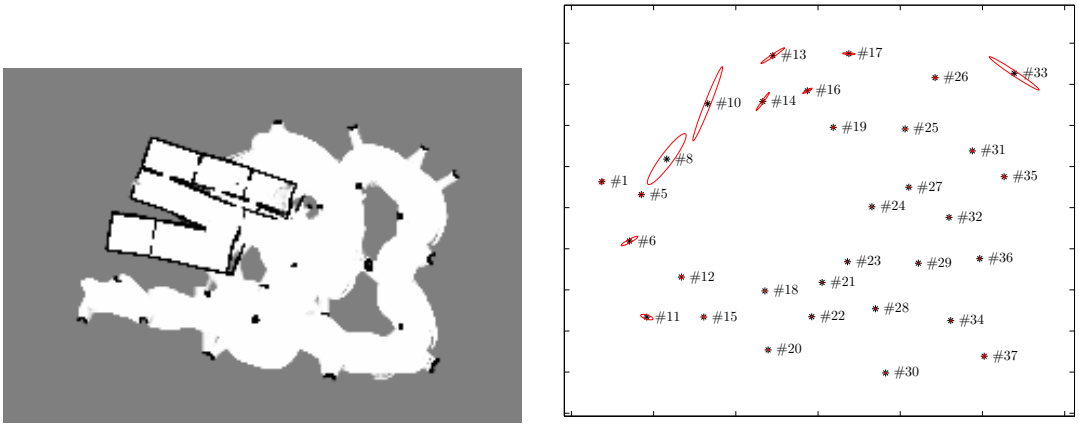
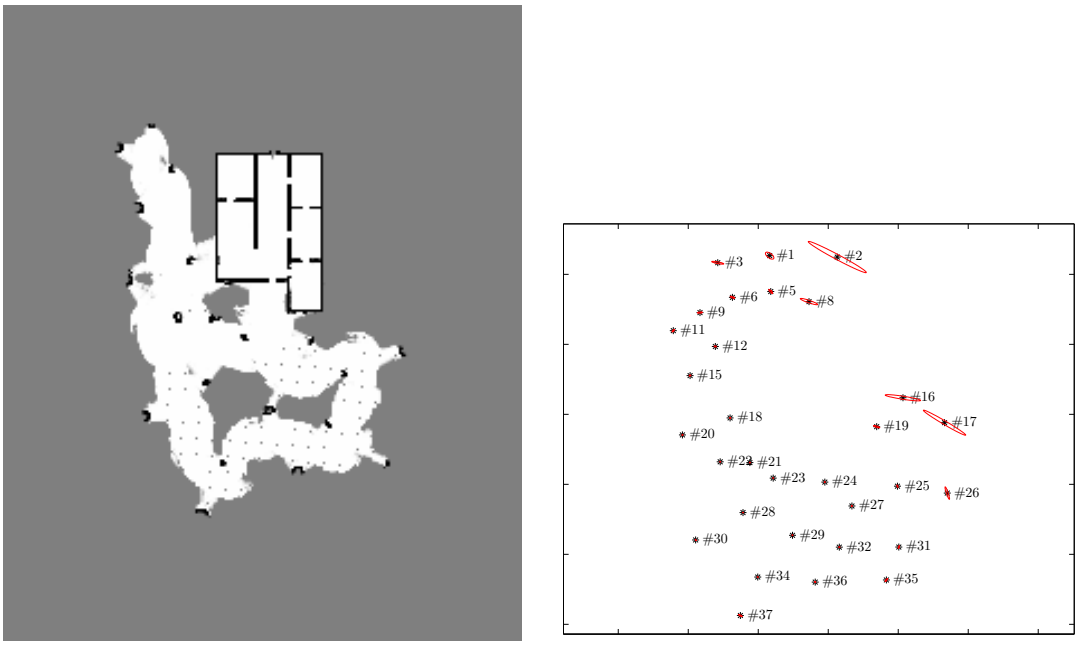


Figure 5.10: Comparison of cumulative error from Single Robot and Multi-Robot SLAM



(a) Occupancy Grid Map (b) Feature Map

Figure 5.11: Sample maps produced during a single robot run



(a) Occupancy Grid Map (b) Feature Map

Figure 5.12: Sample maps produced by two robots that merged midway

Chapter 6

Conclusions

Increasingly, larger environments are required to be mapped and it can be too dangerous or lengthy for humans to accomplish tasks using traditional surveying techniques. Using Multi-Robot SLAM is a suitable solution, but there are still problems that need to be solved. One of these problems is the error incurred when merging maps between two robots.

The goal of the research described in this thesis is to develop a method to reduce the amount of incurred error when merging maps. The difficulty was to determine when to merge the individual robot maps. This goal was achieved and validated, as shown in section 5.2, using simulated data. Using the proposed approach described in chapter 4, reinforcement learning was used to determine which conditions the robot should merge maps to reduce the amount of incurred error, as opposed to merging immediately upon observing another robot. In order to further reduce this incurred error, more accurate means of determining a transformation matrix were proposed. These methods could then be used in conjunction with the method outlined in this thesis to determine when best to merge.

In practice, the resulting policy from this thesis could not be directly applied to every experimental setup. Instead, this proposed approach would need to be

performed using the robot and sensor parameters that matched the available robot setup. Once trained in the desired type of environment, the resulting policy could then be validated using simulated data. The validated policy could then be used in real world experiments.

Reinforcement Learning is shown to be capable of providing a solution to the sub-problem of map merging in the overall SLAM problem. There may be other decision type problems that could be solved using Reinforcement Learning or other machine learning algorithms, such as the determining of particle filter mapping parameters or determination of which sensors should receive more weight depending on the environment.

6.1 Summary of Contributions

In this thesis reinforcement learning was used to determine when a mobile robot should merge maps with another in order to reduce the amount of error. This method was developed and validated using simulated datasets and can be adapted with different parameters for other Multi-Robot scenarios. In summary, this thesis has provided advancements to this area of research through the following contributions:

1. Development of simulated dataset generator: existing simulated dataset generators had fixed parameters sensors, did not have facilities to specify a desired robot trajectory, and environments needed to be specified in the form of pixel images. Therefore, it was necessary to design and develop one for this thesis. In the developed simulator, environments can be simulated using lines to represent indoor walls and circles to represent outdoor features. Sensors and mobile robot parameters are completely customizable. Finally, desired trajectories can be constructed as a series of waypoints which can then be tracked

using a controller.

2. Extension of MRPT C++ library: the MRPT library contained some of the necessary mapping algorithms, but it needed to be extended in some areas to meet the needs of this research. The additional requirements necessary were: a simple two dimensional feature map, map merging of both map types, and observation of other robots. The processing of particle filters was increased through the parallelization of "for loops" that performed the individual particle calculations. MRPT was never designed to be used for Multi-Robot SLAM or the dual representation SLAM from [1] and now has some capabilities to perform this type of SLAM research.

3. Reproduction and validation of dual representation SLAM by Wurm *et al.* [1]: in order to successfully map indoor and outdoor environments this work needed to be reproduced and validated. Wurm's training method was reproduced and the resulting policy was used in the MRPT mapping algorithms implemented for this thesis.

4. Using reinforcement learning in Multi-Robot SLAM: as previously stated, the scope of this thesis was to show that reinforcement learning could be effectively used in Multi-Robot SLAM. This was done by training a policy that could determine in which state and which transformation matrix should be used in order to reduce the amount of incurred error from merging maps from two different robots.

6.2 Future Work

Due to time constraints, certain trials or extensions of this work could not be achieved or properly investigated. Some of these are listed here:

1. Real world experimentation: to further validate of the results shown in this thesis real world experimentation is required. Experimental mobile robots equipped with the necessary sensors would have to be assembled. These robots would also need synchronized data logging software. Finally, a relatively flat and large enough indoor and outdoor area capable of being explored by mobile robots would need to be designed.

2. More Reinforcement States: a higher number of reinforcement learning states causes the necessary training time to increase. Therefore, for this thesis a balance was reached so that every training trial was run for a couple of days. For better and more accurate results, more states could be introduced with a more powerful computer being used.

3. Multiple different training environments: again as a means of increasing the quality of the trained policy, more varied environments could be introduced into the reinforcement learning training. Also, multiple dataset runs for each new environment could be generated. The end result would be a far more diversified training environment.

4. Testing the method with more than two robots: a natural extension would be to investigate how this algorithm performs in a scenario with even more mobile robots. As previously stated, the resulting decision policy is decentralized which means it should be easily extendable to a growing population of robots. Therefore, using existing developed environments datasets with three robots could be

generated and then processed using the trained policy from this thesis.

There still remains plenty of problems requiring solutions in mobile robotics and its subsection of SLAM and thus further research is required. However, the benefits of possible applications of these solutions are numerous. Fleets of robotically controlled vehicles could explore or simply navigate through cities and share their local maps using the algorithm derived in this thesis.

List of References

[1] K. M. Wurm, C. Stachniss, and G. Grisetti, "Bridging the gap between feature- and grid-based slam," *Robot. Auton. Syst.*, vol. 58, pp. 140–148, February 2010. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2009.09.009

[2] R. Smith, M. Self, and P. Cheeseman, *Estimating uncertain spatial relationships in robotics*. New York, NY, USA: Springer-Verlag New York, Inc., 1990, pp. 167–193. [Online]. Available: http://portal.acm.org/citation.cfm?id=93002.93291

[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 476–481.

[4] X. Zhou and S. Roumeliotis, "Multi-robot slam with unknown initial correspondence: The robot rendezvous case," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 1785–1792.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2006.

[6] A. Doucet, J. de Freitas, K. Murphy, and S. Russel, "Rao-blackwellized particle filtering for dynamic bayesian networks," in *Proc. of the Int. Joint conf. on Uncertainty in Artificial Intelligence (UAI)*, 2000, pp. 176–183.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: a factored solution to the simultaneous localization and mapping problem," in *Eighteenth national conference on Artificial intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 593–598. [Online]. Available: http://portal.acm.org/citation.cfm?id=777092.777184

[8] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm

for generating maps of large-scale cyclic environments from raw laser range measurements," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1, 2003, pp. 206–211.

[9] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, "Fastslam 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Proceedings of the 18th international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 1151–1156. [Online]. Available: http://portal.acm.org/citation.cfm?id=1630659.1630824

[10] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 34–46, 2007.

[11] A. Doucet, N. de Freitas, and e. Neil Gordon, *Sequential Mone Carlo methods in practice*. Springer-Verlag, 2000.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998.

[13] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.

[14] N. Ozkucur and H. Akin, "Cooperative multi-robot map merging using fast-slam," in *RoboCup 2009: Robot Soccer World Cup XIII*, vol. 1. Springer Berlin / Heidelberg, 2010, pp. 449–460.

[15] The player project. Open Source. [Online]. Available: http://playerstage.sourceforge.net

[16] Microsoft robotics developer studio 2008 r3. Microsoft. [Online]. Available: http://www.microsoft.com/robotics

[17] Carnegie mellon robot navigation toolkit. Carnegie Mellon. [Online]. Available: http://carmen.sourceforge.net/

[18] Mobile robot programming toolkit. Open Source. [Online]. Available: http://www.mrpt.org

[19] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[20] J. L. Martnez, J. Gonzlez, J. Morales, A. Mandow, and A. J. Garca-Cerezo, "Mobile robot motion estimation by 2d scan matching with genetic and iterative closest point algorithms," *Journal of Field Robotics*, vol. 23, no. 1, pp. 21–34, 2006. [Online]. Available: http://dx.doi.org/10.1002/rob.20104

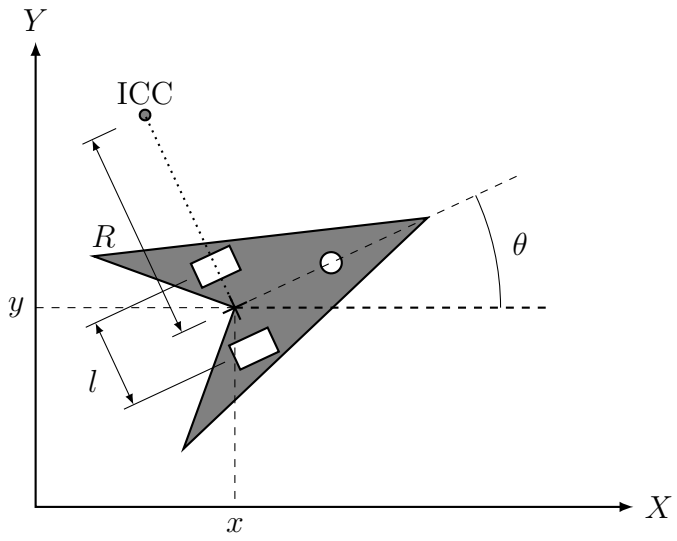
[21] M. Berg, O. Cheong, and M. Kreveld, *Computational geometry: algorithms and applications*. Springer, 2008. [Online]. Available: http://books.google.com/books?id=tkyG8W2163YC

[22] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

Appendix A

Derivation of Differential Drive Robot Controller

The derivation of the kinematic model is based on the following diagram:



where R is the instantaneous curvature radius of the robot trajectory with relation to the center of the axis.

Let v_R and v_L be the linear velocities of the right and left wheel respectively. Analytically, these linear velocities can be determined as

$$v_R = \left(R + \frac{l}{2}\right)\omega \tag{A.1a}$$

$$v_L = \left(R - \frac{l}{2}\right)\omega \tag{A.1b}$$

solving for ω yields

$$\omega = \frac{v_R}{R + \frac{l}{2}} \tag{A.2a}$$

$$\omega = \frac{v_L}{R - \frac{l}{2}} \tag{A.2b}$$

and setting $\omega = \omega$ leads to

$$\frac{v_R}{R+\frac{l}{2}} = \frac{v_L}{R-\frac{l}{2}} \Rightarrow v_R\left(R-\frac{l}{2}\right) = v_L\left(R+\frac{l}{2}\right) \tag{A.3}$$

$$R\left(v_R - v_L\right) = \frac{l}{2}\left(v_R + v_L\right) \tag{A.4}$$

$$\therefore R = \frac{l}{2}\frac{v_R + v_L}{v_R - v_L} \tag{A.5}$$

substituting (A.5) back into (A.2) yields

$$\omega = \frac{v_R}{\left(\frac{l}{2}\frac{v_R+v_L}{v_R-v_L}\right)+\frac{l}{2}} = \frac{v_R}{\frac{l}{2}\left(\frac{v_R+v_L}{v_R-v_L}+\frac{v_R-v_L}{v_R-v_L}\right)} \tag{A.6}$$

$$\therefore \omega = \frac{1}{l}v_R - \frac{1}{l}v_L \tag{A.7}$$

Finally, subsituting (A.5) and (A.7) into $v = R\omega$ yields

$$v = \frac{1}{2}\left(v_R + v_L\right) \tag{A.8}$$

The kinematic equations for the differential drive robot are

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{2}\left(v_R+v_L\right)\right)\cos\theta \\ \left(\frac{1}{2}\left(v_R+v_L\right)\right)\sin\theta \\ \frac{1}{l}v_R - \frac{1}{l}v_L \end{bmatrix} \tag{A.9}$$

$$\therefore \dot{\mathbf{q}} = \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} v_R \\ v_L \end{bmatrix} \tag{A.10}$$

The derivation of the conversion of unicycle input signals to differential drive inputs (3.15) begins by setting up (A.8) and (A.7) into a matrix form

$$\begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{l} & -\frac{1}{l} \\ \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \upsilon_R \\ \\ \upsilon_L \end{bmatrix} \tag{A.11}$$

Now taking the inverse of (A.11) will yield the desired transformation of inputs

$$\begin{bmatrix} \upsilon_R \\ \\ \upsilon_L \end{bmatrix} = \begin{bmatrix} \frac{1}{l} & -\frac{1}{l} \\ \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}^{-1} \begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & \frac{l}{2} \\ \\ 1 & -\frac{l}{2} \end{bmatrix} \begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} \tag{A.12}$$

It is now possible to design a controller assuming that the mobile robot is a unicycle. In order to use exact feedback linearization, the current system dynamics require the introduction of additional states. In any trajectory tracking problem, it is desired that $z - z_d \to 0$ at $t \to \infty$. Where $z = (x, y)^T$. Finding $\dot{z}$ yields

$$\dot{z} = \begin{bmatrix} \dot{x} \\ \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \upsilon \cos\theta \\ \\ \upsilon \sin\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \\ \sin\theta & 0 \end{bmatrix} \begin{bmatrix} \upsilon \\ \\ \omega \end{bmatrix} \tag{A.13}$$

which is not invertible, thus another time derivative is taken

$$\ddot{z} = \begin{bmatrix} \ddot{x} \\ \\ \ddot{y} \end{bmatrix} = \dot{\upsilon} \begin{bmatrix} \cos\theta \\ \\ \sin\theta \end{bmatrix} + \upsilon\dot{\theta} \begin{bmatrix} -\sin\theta \\ \\ \cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\upsilon\sin\theta \\ \\ \sin\theta & \upsilon\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\upsilon} \\ \\ \omega \end{bmatrix} \tag{A.14}$$

where as long as $\upsilon \neq 0$, (A.14) can be inverted. This will lead to the inclusion of new 'state' that is actually one of the unicycle inputs. First, let $\eta := \ddot{z}$ and find the

inverse equation of (A.14)

$$\begin{bmatrix} \dot{\upsilon} \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\dfrac{\sin\theta}{\upsilon} & \dfrac{\cos\theta}{\upsilon} \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \tag{A.15}$$

Finally this can be put into the unicycle model yielding

$$\dot{x} = \upsilon\cos\theta \tag{A.16a}$$

$$\dot{y} = \upsilon\sin\theta \tag{A.16b}$$

$$\dot{\theta} = \omega = -\frac{\sin\theta}{\upsilon}\eta_1 + \frac{\cos\theta}{\upsilon}\eta_2 \tag{A.16c}$$

$$\dot{\upsilon} = \eta_1\cos\theta + \eta_2\sin\theta \tag{A.16d}$$

Now by picking new coordinates $(\zeta_1, \zeta_2, \zeta_3, \zeta_4)^T := (x, y, \dot{x}, \dot{y})^T$, the following linear system can be achieved:

$$\begin{bmatrix} \dot{\zeta}_1 \\ \dot{\zeta}_2 \\ \dot{\zeta}_3 \\ \dot{\zeta}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \tag{A.17}$$

Now that the system has been linearized, a standard linear controller can be designed.

Appendix B

MRPT Code Modifications

B.1 C2DFeature Header File

```
#include <necessary_header_files.h>

namespace mrpt
{
namespace slam
{
  class C2DFeatureMap;

  class MAPS_IMPEXP C2DFeature : public CPointPDF
  {
  public:
    typedef int64_t TFeatureID;

    CPointPDFGaussian m_locationGauss;

    TFeatureID      m_ID;

    C2DFeature();
    virtual ~C2DFeature();

     /** Returns an estimate of the point, (the mean, or mathematical
         expectation of the PDF).
       */
    void getMean(CPoint3D &mean_point) const;

    /** Returns an estimate of the point covariance matrix
        (3x3 cov matrix) and the mean, both at once.
      */
    void getCovarianceAndMean(CMatrixDouble33 &cov,CPoint3D &mean_point) const;

  }; // End of class definition
} // End of namespace
} // End of namespace
```

B.2 C2DFeatureMap Header File

```
#include <necessary_header_files.h>

namespace mrpt
{
namespace slam
{
  class CObservationBearingRange;

  class MAPS_IMPEXP C2DFeatureMap : public CMetricMap
  {
  public:
    typedef std::deque<C2DFeature>  TSequenceFeatures;

  protected:
    /** The individual beacons */
    TSequenceFeatures     m_features;

     virtual bool  internal_insertObservation(
      const CObservation *obs, const CPose3D *robotPose = NULL
      );

  public:
    C2DFeatureMap();

    void push_back(const C2DFeature& m) {
      m_features.push_back( m );
    }

    /** Computes the (logarithmic) likelihood that a given observation
        was taken from a given pose in this map.
     */
    double computeObservationLikelihood(
      const CObservation *obs, const CPose3D &takenFrom
       );

    C2DFeature * getFeatureByID( C2DFeature::TFeatureID  id );

    void  mergeWithOtherMap(
      C2DFeatureMap *other, CPose2D trans
      );

  }; // End of class def.
} // End of namespace
} // End of namespace
```

B.3 C2DFeatureMap Map Merging Code

```
void  C2DFeatureMap::mergeWithOtherMap(
      C2DFeatureMap *other,
      CPose2D     trans
      )
{
  // Transformation values
  double cosine = cos(trans[2]);  double sine   = sin(trans[2]);
  double tx     = trans[0];       double ty     = trans[1];

  CMatrixDouble22 matT;
  matT(0,0) = cosine; matT(0,1) = -sine;
  matT(1,0) = sine; matT(1,1) = cosine;
  C2DFeature* featMatch; double xT, yT;

  for( iterator feat=other->m_features.begin();
       feat!=other->m_features.end(); feat++ )
  {
    // Transform mean
    xT = feat->m_locationGauss.mean.x()*cosine -
         feat->m_locationGauss.mean.y()*sine + tx;
    yT = feat->m_locationGauss.mean.x()*sine +
         feat->m_locationGauss.mean.y()*cosine + ty;

    CMatrixDouble22 curCov,transCov;
    curCov(0,0) = feat->m_locationGauss.cov(0,0);
    curCov(0,1) = feat->m_locationGauss.cov(0,1);
    curCov(1,0) = feat->m_locationGauss.cov(1,0);
    curCov(1,1) = feat->m_locationGauss.cov(1,1);
    matT.multiply_HtCH(curCov,transCov,true,false);
    transCov.force_symmetry();

    featMatch = getFeatureByID(feat->m_ID);

    if( featMatch ) // if the feature already exists
    {
      CMatrixDouble22 curCov, tempMat;
      CMatrixDouble21 mu_incr, diff;

      curCov(0,0) = featMatch->m_locationGauss.cov(0,0);
      curCov(0,1) = featMatch->m_locationGauss.cov(0,1);
      curCov(1,0) = featMatch->m_locationGauss.cov(1,0);
      curCov(1,1) = featMatch->m_locationGauss.cov(1,1);
      diff(0,0) = xT-featMatch->m_locationGauss.mean.x();
      diff(1,0) = yT-featMatch->m_locationGauss.mean.y();
      mrpt::math::detail::invMatrix(curCov+transCov,tempMat);
      tempMat = curCov*tempMat;

      curCov -= tempMat*curCov;
```

```
      featMatch->m_locationGauss.cov(0,0) = curCov(0,0);
      featMatch->m_locationGauss.cov(0,1) = curCov(0,1);
      featMatch->m_locationGauss.cov(1,0) = curCov(1,0);
      featMatch->m_locationGauss.cov(1,1) = curCov(1,1);

      mu_incr = tempMat*diff;
      featMatch->m_locationGauss.mean.x_incr( mu_incr(0,0) );
      featMatch->m_locationGauss.mean.y_incr( mu_incr(1,0) );
    }
    else
    {
      C2DFeature newFeat;
      newFeat.m_ID = feat->m_ID;

      newFeat.m_locationGauss.mean = CPoint3D(xT, yT);

      newFeat.m_locationGauss.cov(0,0) = transCov(0,0);
      newFeat.m_locationGauss.cov(0,1) = transCov(0,1);
      newFeat.m_locationGauss.cov(1,0) = transCov(1,0);
      newFeat.m_locationGauss.cov(1,1) = transCov(1,1);

      // and insert it:
      m_features.push_back( newFeat );
    }
  }
}
```

B.4 COccupancyGridMap2D Map Merging Code

```
void  COccupancyGridMap2D::mergeWithOtherMap(
      COccupancyGridMap2D *other,
      CPose2D         trans
      )
{
  // Transformation values
  double cosine = cos(trans[2]);  double sine   = sin(trans[2]);
  double tx     = trans[0];       double ty     = trans[1];

  { // First need to resize grid appropriately
    double P1Tx = other->x_min*cosine-other->y_min*sine+tx;
    double P1Ty = other->x_min*sine+other->y_min*cosine+ty;
    double P2Tx = other->x_max*cosine-other->y_min*sine+tx;
    double P2Ty = other->x_max*sine+other->y_min*cosine+ty;
    double P3Tx = other->x_min*cosine-other->y_max*sine+tx;
    double P3Ty = other->x_min*sine+other->y_max*cosine+ty;
    double P4Tx = other->x_max*cosine-other->y_max*sine+tx;
    double P4Ty = other->x_max*sine+other->y_max*cosine+ty;
    double rXmin = min<double>(min(min(P1Tx,P2Tx),min(P3Tx,P4Tx)),this->x_min);
    double rYmin = min<double>(min(min(P1Ty,P2Ty),min(P3Ty,P4Ty)),this->y_min);
    double rXmax = max<double>(max(max(P1Tx,P2Tx),max(P3Tx,P4Tx)),this->x_max);
    double rYmax = max<double>(max(max(P1Ty,P2Ty),max(P3Ty,P4Ty)),this->y_max);
    this->resizeGrid(rXmin,rXmax,rYmin,rYmax);
  }

  int xT, yT, xi, yi;
  double x, y;
  int temp;
  // Now simply need to go through each cell in the other grid, transform it
  for( yi=0, y=other->y_min; yi<static_cast<int>(other->size_y);
      yi++, y+=other->resolution )
  {
    for( xi=0, x=other->x_min; xi<static_cast<int>(other->size_x);
      xi++,x+=other->resolution )
    {
      // check value to ensure it's not zero
      if( other->map[xi+yi*other->size_x] != 0.0 )
      {
        xT = static_cast<int>((x*cosine-y*sine+tx-x_min)/resolution-0.5f );
        yT = static_cast<int>((x*sine+y*cosine+ty-y_min)/resolution-0.5f );

        temp = map[xT+yT*size_x] + other->map[xi+yi*other->size_x];
        map[xT+yT*size_x] =
          max(min(OCCGRID_CELLTYPE_MAX,temp),OCCGRID_CELLTYPE_MIN);
      }
    }
  }
}
```

B.5 CObservationOtherRobot Header File

```
#include <necessary_header_files.h>

namespace mrpt
{
namespace slam
{
  class OBS_IMPEXP CObservationOtherRobot : public CObservation
  {
   public:
    CObservationOtherRobot( );

    float minSensorDistance, maxSensorDistance;  // Ranges, in meters
    float fieldOfView_yaw;    //The "field-of-view" of the sensor, in radians
    float fieldOfView_pitch;  //The "field-of-view" of the sensor, in radians

    // The position of the sensor on the robot.
    CPose3D   sensorLocationOnRobot;

    struct OBS_IMPEXP TMeasurement
    {
      float range;  // The sensed landmark distance, in meters.
      float yaw,pitch;  // The sensed landmark direction, in radians
      int32_t robotID;  // The ID of the sensed robot
    };

    typedef std::vector<TMeasurement> TMeasurementList;
    TMeasurementList sensedData;

    float sensor_std_range,sensor_std_yaw,sensor_std_pitch;

    void getSensorPose( CPose3D &out_sensorPose ) const
    { out_sensorPose = sensorLocationOnRobot; }
    void setSensorPose( const CPose3D &newSensorPose )
    { sensorLocationOnRobot = newSensorPose; }
  }; // End of class def.
} // End of namespace
} // End of namespace
```

Appendix C

MRPT Particle Filter Parameters

C.1 Config File

```
//=======================================================
// Section: [MappingApplication]
// Use: Here comes global parameters for the app.
//=======================================================

// The directory where the log files will be saved
// (left in blank if no log is required)
logOutput_dir=
LOG_FREQUENCY=1     // The frequency of log files generation:
GENERATE_LOG_JOINT_H=0
GENERATE_LOG_INFO=0
SAVE_MAP_IMAGES=1
SAVE_3D_SCENE=0
SAVE_POSE_LOG=0
SAVE_ENOSE_READINGS=0
CAMERA_3DSCENE_FOLLOWS_ROBOT=0
SHOW_PROGRESS_IN_WINDOW=0

// The distance threshold for inserting observations in the map (meters)
insertionLinDistance=0
// The distance threshold for inserting observations in the map (degrees)
insertionAngDistance_deg=0

// ------------------------------------------------------------------
// The Particle Filter algorithm:
//  0: pfStandardProposal
//  1: pfAuxiliaryPFStandard
//  2: pfOptimalProposal      *** (ICP-based (Grisetti's method),...)
//  3: pfAuxiliaryPFOptimal   *** (Optimal SAMPLING)
// ------------------------------------------------------------------
PF_algorithm    = 2
adaptiveSampleSize  = 0   // 0: Fixed # of particles, 1: KLD adaptive

// ------------------------------------------------------------------
// The Particle Filter Resampling method:
//  0: prMultinomial
//  1: prResidual
//  2: prStratified
//  3: prSystematic
// ------------------------------------------------------------------
resamplingMethod=0

sampleSize=25     // Sample size (for fixed number)
BETA=0.50     // Resampling ESS threshold
```

```
// ====================================================
//            MULTIMETRIC MAP CONFIGURATION
// ====================================================
// Creation of maps:
occupancyGrid_count=1
gasGrid_count=0
landmarksMap_count=0
beaconMap_count=0
pointsMap_count=0
2dfeatureMap_count=1

// Selection of map for likelihood:
// (fuseAll=-1,occGrid=0, points=1,landmarks=2,gasGrid=3)
likelihoodMapSelection=-1


// ====================================================
//         CHybridMetricMapPDF::TPredictionParams
// ====================================================
powFactor=1         // A "power factor" for updating weights
pfAuxFilterOptimal_MaximumSearchSamples=250 // For PF algorithm=3

// ------------------------------------------------------------------
//   pfOptimalProposal_mapSelection
//  Only for PF algorithm=2 (Exact "pfOptimalProposal")
//   Select the map on which to calculate the optimal
//    proposal distribution. Values:
//   0: Gridmap   -> Uses Scan matching-based approximation
//   1: Landmarks -> Uses matching to approximate optimal
//   2: Beacons   -> Used for exact optimal proposal in RO-SLAM
// ------------------------------------------------------------------
pfOptimalProposal_mapSelection=-1
bDualSlam=1
verbose=0

// ====================================================
//   MULTIMETRIC MAP: BeaconMap #00
// ====================================================
// Creation Options for BeaconMap 00:
[MappingApplication_beaconMap_00_insertOpts]
disableSaveAs3DObject=0
instertOnlyOneObsv=0

// Likelihood Options for BeaconMap 00:
[MappingApplication_beaconMap_00_likelihoodOpts]
rangeStd=0.1
bearingStd=0.1
```

```
// ======================================================
//   MULTIMETRIC MAP: OccGrid #00
// ======================================================
// Creation Options for OccupancyGridMap 00:
[MappingApplication_occupancyGrid_00_creationOpts]
resolution=0.25
disableSaveAs3DObject=0


// Insertion Options for OccupancyGridMap 00:
[MappingApplication_occupancyGrid_00_insertOpts]
mapAltitude         = 0
useMapAltitude        = 0
maxOccupancyUpdateCertainty = 0.51
considerInvalidRangesAsFreeSpace = 1
minLaserScanNoiseStd    = 0.001
wideningBeamsWithDistance   = 1


// Likelihood Options for OccupancyGridMap 00:
[MappingApplication_occupancyGrid_00_likelihoodOpts]
// 0=MI, 1=Beam Model, 2=RSLC, 3=Cells Difs, 4=LF_Trun, 5=LF_II
likelihoodMethod=4

LF_decimation=5
LF_stdHit=0.20
LF_maxCorrsDistance=0.30
LF_zHit=0.999
LF_zRandom=0.001
LF_alternateAverageMethod=0

enableLikelihoodCache=1
```