Real Time SLAM Using Compressed Occupancy Grids For a Low Cost Autonomous Underwater Vehicle

Christopher H. Cain

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Alexander Leonessa, Chair
John B. Ferris
Dennis W. Hong
Andrew J. Kurdila
Craig A. Woolsey

February 25, 2014
Blacksburg, Virginia

Keywords: Autonomous Vehicles, SLAM, Occupancy Grids, Haar Wavelet Transform, Compressed Sensing, Sparse Signal Reconstruction, Data Compression

Real Time SLAM Using Compressed Occupancy Grids For a Low Cost Autonomous Underwater Vehicle

Christopher H. Cain

ABSTRACT

The research presented in this dissertation pertains to the development of a real time SLAM solution that can be performed by a low cost autonomous underwater vehicle equipped with low cost and memory constrained computing resources. The design of a custom rangefinder for underwater applications is presented. The rangefinder makes use of two laser line generators and a camera to measure the unknown distance to objects in an underwater environment. A visual odometry algorithm is introduced that makes use of a downward facing camera to provide our underwater vehicle with localization information. The sensor suite composed of the laser rangefinder, downward facing camera, and a digital compass are verified, using the Extended Kalman Filter based solution to the SLAM problem along with the particle filter based solution known as FastSLAM, to ensure that they provide information that is accurate enough to solve the SLAM problem for out low cost underwater vehicle. Next, an extension of the FastSLAM algorithm is presented that stores the map of the environment using an occupancy grid is introduced. The use of occupancy grids greatly increases the amount of memory required to perform the algorithm so a version of the FastSLAM algorithm that stores the occupancy grids using the Haar wavelet representation is presented. Finally, a form of the FastSLAM algorithm is presented that stores the occupancy grid in compressed form to reduce the amount memory required to perform the algorithm. It is shown in experimental results that the same result can be achieved, as that produced by the algorithm that stores the complete occupancy grid, using only 40% of the memory required to store the complete occupancy grid.

Contents

1 Introduction **1**

1.1. The Autonomous Robotic Pool Cleaner 2

1.1.1. Intelligent Pool Cleaning 5

1.1.2. User Free Operation 7

1.1.3. ARPC Prototype 7

1.2. Contribution 10

1.2.1. Hardware 10

1.2.2. Software 10

1.3. Outline 11

2 Preliminary Material **12**

2.1. Literature Review 12

2.1.1. Laser Range Finder 12

2.1.2. Visual Odometry 13

2.1.3. Gaussian SLAM 15

2.1.4. Non-Parametric SLAM 17

2.1.5. Feature Extraction 18

2.1.6. Underwater SLAM 19

2.1.7. Sparse Signal Recovery 20

2.2. Mathematical Preliminaries 20

2.2.1. Notation 20

2.2.2. Probability Concepts 21

2.2.3. The SLAM Problem as a Recursive Bayesian Estimation Problem . . 23

3 Camera and Laser Rangefinder **26**

3.1. Introduction 26

3.2. Physical Layout . . . 27
3.3. Pinhole Camera Model . . . 28
3.4. Distance Measurement Theory . . . 31
3.5. Image Processing Algorithm . . . 32
3.5.1. Distortion Removal . . . 32
3.5.2. Image Segmentation . . . 34
3.5.3. Line Extraction . . . 35
3.6. Experimental Results . . . 37
3.6.1. In Air Testing . . . 37
3.6.2. Underwater Testing . . . 39
3.7. Error Analysis . . . 40
3.8. Conclusions . . . 41

4 Visual Odometry with Downward Facing Camera 42
4.1. Introduction . . . 42
4.2. Visual Odometry Algorithm . . . 44
4.3. Experimental Results . . . 48
4.4. Conclusions . . . 51

5 Sensor Validation with EKF SLAM 52
5.1. Introduction . . . 52
5.2. The SLAM Problem . . . 53
5.3. EKF SLAM Algorithm . . . 53
5.3.1. Prediction . . . 55
5.3.2. Correction . . . 57
5.3.3. Augmentation . . . 58
5.4. Feature Extraction and Data Association . . . 59
5.5. Experimental Results . . . 61
5.6. Conclusions . . . 64

6 Sensor Validation with FastSLAM 67
6.1. Introduction . . . 67
6.2. FastSLAM Algorithm . . . 68
6.2.1. Pose Sampling . . . 69
6.2.2. Feature Location Estimation . . . 72

6.2.3. Importance Weight Calculation . . . 73
6.2.4. Resampling . . . 75
6.3. New Feature Addition . . . 75
6.4. Feature Extraction and Data Association . . . 75
6.5. Experimental Results . . . 76
6.6. Conclusions . . . 77

7 FastSLAM With Occupancy Grids 82
7.1. Introduction . . . 82
7.2. Occupancy Grid Maps . . . 83
7.3. FastSLAM with Occupancy Grids Algorithm . . . 88
7.3.1. Pose Sampling . . . 89
7.3.2. Importance Weight Calculation . . . 91
7.3.3. Occupancy Grid Update . . . 92
7.3.4. Resampling . . . 92
7.4. Experimental Results . . . 93
7.5. Conclusions . . . 94

8 Wavelet Representation of Occupancy Grids For FastSLAM Implementation 99
8.1. Introduction . . . 99
8.2. FastSLAM With Occupancy Grids Stored Using an Alternate Basis . . . 100
8.2.1. Alternate Occupancy Grid Representation . . . 101
8.2.2. Pose Sampling . . . 103
8.2.3. Importance Weight Calculation . . . 104
8.2.4. Occupancy Grid Update . . . 106
8.2.5. Resampling . . . 107
8.3. Haar Wavelet Basis . . . 108
8.3.1. Haar Wavelet Transform . . . 108
8.3.2. Optimized Haar Wavelet Transform . . . 110
8.4. Experimental Results . . . 113
8.4.1. Results using LiDAR . . . 113
8.4.2. Results Using Laser Based Rangefinder . . . 115
8.5. Conclusions . . . 120

9 FastSLAM With Compressed Occupancy Grids 125

9.1. Introduction . . . 125
9.2. FastSLAM With Compressed Occupancy Grids . . . 126
9.2.1. Compressed Occupancy Grid Representation . . . 129
9.2.2. Pose Sampling . . . 130
9.2.3. Importance Weight Calculation . . . 131
9.2.4. Compressed Occupancy Grid Update . . . 132
9.2.5. Resampling . . . 134
9.3. Compressed Signal Reconstruction Methods . . . 135
9.3.1. l_2 Reconstruction (l_2) . . . 136
9.3.2. Compressed Sensing Reconstruction (CS) . . . 139
9.3.3. Compressed Sensing Embedded Kalman Filtering (CSEKF) . . . 140
9.3.4. Kalman Filter Constrained with Quasi-Norm (CSEKF-p) . . . 143
9.3.5. Algorithm Comparison . . . 147
9.4. Performance Improvements . . . 151
9.4.1. Updated Compression Step . . . 152
9.4.2. Updated l_2 reconstruction . . . 156
9.5. Experimental Results . . . 159
9.5.1. Results Using LiDAR . . . 160
9.5.2. Results Using Laser Based Rangefinder . . . 166
9.5.3. How much compression can be achieved? . . . 170
9.6. Conclusions . . . 176

10 Conclusions and Future Research **178**

Bibliography **181**

A EKF SLAM Details **189**
A.1. Introduction . . . 189
A.2. Model Verification . . . 190
A.2.1. State Transition Model . . . 190
A.2.2. Measurement Model . . . 194
A.3. Prediction . . . 195
A.3.1. Mean Vector Prediction . . . 196
A.3.2. Covariance Matrix Prediction . . . 197
A.4. Correction . . . 198

A.4.1. Mean Vector Correction 198
A.4.2. Covariance Matrix Correction 199
A.5. State Augmentation 199
A.5.1. Mean Vector Augmentation 200
A.5.2. Covariance Matrix Augmentation 201

B FastSLAM Details **203**
B.1. Introduction 203
B.2. Pose Sampling 204
B.3. Landmark Estimate Update 208
B.3.1. Prediction Phase 209
B.3.2. Correction 209
B.4. Importance Weight Calculation 211
B.5. Resampling 212
B.6. New Feature Addition 213

C FastSLAM OG Details **215**
C.1. Introduction 215
C.1.1. Pose Sampling 216
C.2. Importance Weight Calculation 217
C.3. Occupancy Grid Map Update 220
C.3.1. Inverse Sensor Model 221
C.4. Resampling 222

List of Figures

1.1 Examples of current automatic pool cleaners sold by Pentair Pool and Spa®. 4
1.2 Initial ARPC prototype developed by VT senior design team. 9
1.3 Second ARPC prototype with the vibrating propulsion motors (a), suction pumps (b), filter cartridge (c), and skimming device (d). 9

3.1 Design of the prototype sensor on the left and the actual prototype on the right. 28
3.2 Absorption coefficient of light in water as a function of wavelength over the total spectrum on the left and the visible spectrum on the right (obtained using data from [1]). 29
3.3 The classic pinhole model of a camera. 30
3.4 Projections of the classic pinhole camera model in the xz plane (a) and the yz plane (b). 30
3.5 Projections of the modified pinhole camera model in the xz plane (a) and the yz plane (b). 31
3.6 Side view of the sensor 32
3.7 Overview of the distance calculation algorithm. 33
3.8 The segmentation process applied to a single video frame. 35
3.9 Overview of the Line Extraction Algorithm. 35
3.10 Each step of the laser line extraction algorithm. 37
3.11 Sensor prototype results vs. LiDAR over time at a relative bearing of $-8°$ (a), $0°$ (b), and $8°$ (c). 38
3.12 Sensor error as a function of distance at a relative bearing of $-8°$ (a), $0°$ (b), and $8°$ (c). 38
3.13 Comparison of a corner as viewed by the prototype sensor and LiDAR from $4.5m$ (a) and $1.5m$ (b). 39
3.14 Experimental results when measuring distances underwater. 40

4.1 Overview of visual odometry algorithm. 45
4.2 Each preprocessing step in the visual odometry algorithm. 45

4.3 The location of the template image (a), the extracted template used for tracking purposes (b), and the image that was searched (c). 47
4.4 The cross correlation matrix $\mathbf{C}_k$ generated from the template matching step. 47
4.5 Overview of the visual odometry model. 49
4.6 Test platform used for indoor tests. 50
4.7 Visual odometry path estimate (a) and the error in the estimate as a function of time (b). 50

5.1 Corner identification using the modified RANSAC algorithm. 60
5.2 The EKF SLAM produced path and map estimate (a) and the error in the position estimate over time (b). 62
5.3 An evenly spaced sequence of images through out the entire localization and mapping process. 63
5.4 The error in the EKF SLAM position estimate in the horizontal (a) and vertical direction (b) over time along with the 2σ boundary. 64
5.5 The error in the EKF SLAM horizontal landmark position estimate over time along with the 2σ boundary. 65
5.6 The error in the EKF SLAM vertical landmark position estimate over time along with the 2σ boundary. 66

6.1 The FastSLAM produced path and map estimate (a) and the error in the position estimate over time (b). 77
6.2 Evenly spaced sequence of path and map estimate produced by the FastSLAM algorithm. 78
6.3 The error in the FastSLAM position estimate in the horizontal (a) and vertical direction (b) over time along with the 2σ boundary. 79
6.4 The error in the FastSLAM horizontal landmark position estimate over time along with the 2σ boundary. 80
6.5 The error in the FastSLAM vertical landmark position estimate over time along with the 2σ boundary. 81

7.1 Example of an unknown environment in which an unmanned vehicle operates (a) the grid that is overlaid on the environment to represent the world (b). . 85
7.2 The FastSLAM OG produced path estimate (a) and map (b). 93
7.3 The error in the FastSLAM OG position estimate over time. 94
7.4 Evenly spaced sequence of path estimates produced by the FastSLAM OG algorithm. 95

7.5 Evenly spaced sequence of map estimates produced by the FastSLAM OG algorithm. 96
7.6 Comparison of the FastSLAM OG position error to the VO position error. . 97
7.7 Comparison of the true occupancy grid (a), the FastSLAM OG occupancy grid (b), and VO occupancy grid (c). 97

8.1 The scaling functions that serve as the basis for V^1. 109
8.2 The wavelet functions that serve as the basis for W^2. 111
8.3 The FastSLAM OG produced path estimate (a) and map (b) that serve as a baseline generated with LiDAR. 114
8.4 The FastSLAM AOG produced path estimate (a) and map (b) generated with LiDAR. 115
8.5 Evenly spaced sequence of path estimates produced by the FastSLAM AOG algorithm generated with LiDAR. 116
8.6 Evenly spaced sequence of map estimates produced by the FastSLAM AOG algorithm generated with LiDAR. 117
8.7 Evenly spaced sequence of occupancy grid coefficient sets produced by the FastSLAM AOG algorithm generated with LiDAR. 118
8.8 The error in the FastSLAM AOG position estimate over time generated with LiDAR. 119
8.9 The FastSLAM OG produced path estimate (a) and map (b) that serves as a baseline generated with custom laser rangefinder. 119
8.10 The FastSLAM AOG produced path estimate (a) and map (b) generated with custom laser rangefinder. 120
8.11 Evenly spaced sequence of path estimates produced by the FastSLAM AOG algorithm generated with custom laser rangefinder. 121
8.12 Evenly spaced sequence of map estimates produced by the FastSLAM AOG algorithm generated with custom laser rangefinder. 122
8.13 Evenly spaced sequence of occupancy grid coefficient sets produced by the FastSLAM AOG algorithm generated with custom laser rangefinder. 123
8.14 The error in the FastSLAM OG position estimate over time generated with custom laser rangefinder. 124

9.1 (a) The test function given by (9.19) and (b) the test function represented in the discrete cosine basis. 136
9.2 The test function reconstructed by minimizing the l_2 error for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data. 138

9.3 The test function reconstructed by solving the compressed sensing convex optimization problem for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data. 140
9.4 The test function reconstructed using the compressed sensing embedded Kalman Filter for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data. 145
9.5 The test function reconstructed using the quasi-norm constrained Kalman Filter for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data. 149
9.6 MSE for different values of p for a single compressed percentage 75%. 150
9.7 MSE as a function of signal data used for each of the four reconstruction algorithms. 150
9.8 Run-time as a function of compressed percentage for each of the four reconstruction algorithms. 151
9.9 An example occupancy grid of an unknown environment (a) and the coefficients that represent it using Haar wavelet basis (b). 155
9.10 Comparison of the reconstruction performance of the normalized Hadamard compression matrix and the matrix that selects the m most significant coefficients. 156
9.11 The comparison in run time for the original l_2 reconstruction along with the updated form of the reconstruction algorithm. 159
9.12 The FastSLAM OG produced path estimate (a) and map (b) used as a baseline generated with LiDAR. 161
9.13 The FastSLAM COG produced path estimate (a) and map (b) generated using the l_2 reconstruction approach with 40% of the data and LiDAR. 162
9.14 The FastSLAM COG produced path estimate (a) and map (b) generated using the compressed sensing reconstruction approach with 40% of the data and LiDAR. 162
9.15 The MSE as a function of data percentage for the estimated path (a) and map (b) generated with LiDAR. 163
9.16 A series of final occupancy grids generated by the FastSLAM COG algorithm using the l_2 reconstruction approach and LiDAR data using 75% (a), 70% (b), 60% (c), 50% (d), 40% (e), 30% (f), 25% (g), 20% (h), and 10% (i) of the full amount of data. 164
9.17 A series of final occupancy grids generated by the FastSLAM COG algorithm using the compressed sensing reconstruction approach and LiDAR data using 75% (a), 70% (b), 60% (c), 50% (d), 40% (e), 30% (f), 25% (g), 20% (h), and 10% (i) of the full amount of data. 165

9.18 The average time to complete a single iteration of the FastSLAM COG algorithm as a function of data used generated with LiDAR data. 166
9.19 The FastSLAM OG produced path estimate (a) and map (b) used as a baseline generated with custom laser rangefinder. 167
9.20 The FastSLAM COG produced path estimate (a) and map (b) generated using the l_2 reconstruction approach with 40% of the data and custom laser rangefinder. 168
9.21 The FastSLAM COG produced path estimate (a) and map (b) generated using the compressed sensing reconstruction approach with 40% of the data and custom laser rangefinder. 168
9.22 The MSE as a function of data percentage for the estimated path (a) and map (b) generated with custom laser rangefinder. 169
9.23 The average time to complete a single iteration of the FastSLAM COG algorithm as a function of data used generated with custom laser rangefinder data. 169
9.24 Occupancy grid of simple rectangular environment using 100% of the data and a cell size of $0.1m$. 171
9.25 MSE between the true occupancy grid of the simple environment and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes. 171
9.26 Occupancy grid of rectangular environment with complex corners using 100% of the data and a cell size of $0.1m$. 172
9.27 MSE between the true occupancy grid of the environment with medium complexity and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes. 173
9.28 Occupancy grid of the complex environment using 100% of the data and a cell size of $0.1m$. 173
9.29 MSE between the true occupancy grid of the complex environment and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes. 174
9.30 MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 1.0m. 174
9.31 MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.5m. 175
9.32 MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.2m. 175
9.33 MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.1m. 176

A.1 Measurement model of the system in the vehicle's local frame (a) and in the worlds global frame (b). 196

C.1 Beam measurement model used by the laser based rangefinder used for the importance weight update. 220
C.2 Inverse sensor model used for the developed laser based rangefinder. 223

List of Algorithms

5.1 Overview of the feature based EKF SLAM algorithm. 56
6.1 Overview of the feature based FastSLAM algorithm. 70
6.2 Overview of the feature based FastSLAM algorithm Part2. 71
7.1 Overview of the standard occupancy grid update algorithm. 87
7.2 Overview of the FastSLAM OG algorithm 90
8.1 Overview of the FastSLAM AOG algorithm 102
8.2 Construction of the occupancy grid update vector $\boldsymbol{\alpha}_k^{[p]}$. 107
8.3 Recursive Haar wavelet decomposition from [2]. 112
9.1 Overview of the FastSLAM COG algorithm 128
9.2 Construction of the occupancy grid update vector $\boldsymbol{\alpha}_k^{[p]}$. 133
9.3 Overview of the CSEKF reconstruction algorithm. 144
9.4 Overview of CSEKF-p reconstruction algorithm. 148
9.5 Overview of the updated compression algorithm that selects the m coefficients with largest magnitudes. 155
9.6 Overview of the optimized l_2 reconstruction algorithm. 158
A.1 Overview of the feature based EKF SLAM algorithm repeated for reference. 191
B.1 Overview of the feature based FastSLAM algorithm repeated for reference. . 205
B.2 Overview of the feature based FastSLAM algorithm repeated for reference Part2. 206
B.3 Overview of the algorithm to sample from a Gaussian distribution. 207
B.4 Overview of the low variance sampler algorithm. 212
C.1 Overview of the FastSLAM OG algorithm repeated for reference. 216
C.2 Overview of the standard occupancy grid update algorithm repeated for reference. 221

List of Tables

4.1 7×7 Laplacian Approximation Kernel 46

Chapter 1

Introduction

The use of robotic vehicles to perform tasks for humans has become an area of intense interest recently and their potential areas of use are greatly varied. They can perform tasks that are deemed too dangerous for humans such as bomb disposal or flying military missions in dangerous environments. They can perform tasks more cheaply than their human counterparts in industrial applications such as warehouse management. A further area of interest is that robotic vehicle can perform repetitive tasks that are tedious for the operator, such as indoor vacuum cleaning or gutter cleaning.

The level of intelligence possessed by these robotic vehicles can vary greatly. Some robotic vehicles such as bomb disposal vehicles or unmanned military drones possess little or no on board intelligence. These vehicles tend to be controlled by operators at remote locations. Many popular robotic vacuum cleaners possess some intelligence ([3]) as they clean by performing pseudo random routines while using on board sensors to avoid walls or furniture and to avoid falling off of stairs. Warehouse robots have higher levels of intelligence that allows them to reach desired locations to retrieve products and fulfill orders without human intervention. However, this intelligence is obtained by modifying the environment in which the robot is operating. In many cases sensors are positioned in the environment allowing robots to know where they are and they possess a map that allows them to navigate.

An area of continued research for robotic applications is the development of sensors and algorithms that allow robots to operate completely autonomously. In order to be truly autonomous an unmanned vehicle must be able to placed into any unknown environment without any knowledge of its surroundings. Using only preceptive sensors an autonomous vehicle should be able to determine where it is in an environment along with the locations of objects that surround it. In the robotics literature this is referred to as the Simultaneous Localization And Mapping (SLAM) problem. Many applications of SLAM involve the use of expensive sensors or powerful computing resources. The research presented in this dissertation provides a full solution to the SLAM problem, including low cost hardware and the required algorithms, that can be performed by an inexpensive unmanned underwater vehicle, equipped with low cost computer resources, in real time.

1.1. The Autonomous Robotic Pool Cleaner

The motivation for this research comes from a project funded by Pentair Pool and Spa®(Pentair). Design engineers at Pentair approached researchers at Virginia Tech looking for collaborators in the development of a fully autonomous pool cleaner for residential sized pools. The motivation for this product comes from the view of Pentair that there is a void in the market for a product of this type. Currently there are several families of automatic pool cleaners available in the market. The first family of cleaners are known as suction side pool cleaners. Suction side cleaners are connected to the input of the pool's filter system, either through the skimmer intake or a dedicated suction port. The *suction* of water through these cleaners forces them to move through the pool in a random pattern while debris is transported from the cleaner to the filter system, an example of this style of cleaner is Pentair's Kreepy Krauly® Kruiser™ (Figure 1.1a). A second family of automatic pool cleaners, known as pressure side pool cleaners, work similar to suction side cleaners except they are

attached to the return line of the filter system rather than the inlet to the system. Pressure side cleaners use the force of the water being returned from the filter system, sometimes boosted with a pump, to provide a propulsive force that moves the cleaner about the pool. As with the suction side cleaners the movement of the pressure side cleaners through the pool is random and these cleaners store the collected debris in an attached filter that must then be emptied by the user once the cleaning process is complete. An example of a pressure side cleaner is Pentair's Kreepy Krauly® Legend® II (Figure 1.1b). The final family of automatic cleaners currently on the market are known as robotic cleaners. These cleaners are typically wheeled vehicles with the power for the drive motors being supplied through an electrical umbilical connected to the surface. The motion of these cleaners is similar to the other families of automatic pool cleaners in that the movement though the pool is pseudo random. However, as opposed to the previous families where the random movement was due to the geometry of the pool and cleaner in robotic cleaners one of several random patterns is performed by the cleaner. These cleaning patterns tend to be for a set period of time over which the robotic cleaner attempts to clean the entire pool. As with the pressure side family of cleaners, robotic cleaners store collected debris in an attached container which is emptied by the operator once the cleaning pattern has been completed. An example of a robotic pool cleaner is Pentair's Kreepy Krauly® Prowler® 710 (Figure 1.1c).

While automatic pool cleaners are becoming more popular in the market there are several downfalls in the currently available technology, specifically with robotic cleaners, that Pentair has observed and hopes to overcome with the Autonomous Robotic Pool Cleaner (ARPC). The primary disadvantage that the ARPC is designed to overcome is the method in which current robotic cleaners clean the pool. In current cleaners a cleaning pattern is performed over a certain period of time. However, due to the lack of intelligence possessed by the cleaner there is no way for the cleaner to ensure that the entire pool has been cleaned once

(a) (b) (c)

Figure 1.1: Examples of current automatic pool cleaners sold by Pentair Pool and Spa®.

the pattern is finished. The result is that in many cases there are portions of the pool that have not been cleaned while other areas of the pool might have been covered by the cleaner multiple times. The second downfall of current robotic cleaners that Pentair has identified through their market research is the amount of user effort required to operate a robotic pool cleaner. In order to operate current cleaners the user first retrieves the cleaner from storage, attaches the power supply to the cleaner, and finally places the cleaner in the pool before manually starting the cleaning process. The cleaning process is then manually started by the user. The cleaner can operate unattended, however it is a good idea that the user monitors the cleaner to ensure that it is operating correctly. There are two issues that commonly occur during the cleaning operation that the operator must fix. First, due to the use of an external power source to provide the power required by the motors and pumps there is a chance that the cleaner can become entangled during the cleaning operation. This entanglement occurs when the attached power cord gets snagged on some component of the pool's geometry and

prevents the cleaner from moving. The second issue that can arise during operation is the onboard debris storage can become full. When the storage container becomes full the cleaner can no longer properly clean. When this occurs the operator must remove the cleaner from the pool to empty the debris storage before placing the cleaner back in the pool to continue cleaning.

1.1.1. Intelligent Pool Cleaning

It was previously stated that one of the primary disadvantages to current robotic pool cleaners is their inability to ensure that the entire pool is cleaned during a given cleaning period. In order to compensate for this the primary design goal for the ARPC is to develop a system that uses attached sensors and intelligence to clean the entire pool, while this is a lofty goal itself Pentair engineers would like to take the idea a step further. However, Pentair would like to take this a step further. Rather than just ensuring that the cleaner cleans the entire pool during each cleaning operation they would like the cleaner to clean the pool in an *intelligent* manner. This intelligent cleaning can be thought of as the ARPC only cleaning those areas of the pool that need to be cleaned during each cleaning operation, that is when the cleaner is placed in the pool for a cleaning operation the cleaner will have the intelligence to know what areas of the pool are dirty at that given time and the cleaner will only go to those areas of the pool and clean them.

In order to perform this intelligent cleaning several system must be developed. First, and the primary goal of this research, a system of sensors and algorithms must be developed that allow the ARPC to know its location in the pool at any given time and since each pool is unique the system must allow the ARPC to learn the layout of the pool in which it is operating. The ARPC must then possess the ability to determine it's location in the learned pool layout. In a robotics sense this is the Simultaneous Localization and Mapping (SLAM)

problem that must be overcome by the ARPC. A key aspect of the ARPC's SLAM solution is its ability to run on low cost embedded computing resources. As the goal of the project is to develop a consumer product the size and cost of the onboard computing resources is an important factor so the developed solution must be able to be performed using low cost embedded computing hardware.

Once the system has been developed that allows the ARPC to know where it is and what is around it the second system that must be developed is one that allows the ARPC to know what areas of the pool are dirty at the current moment so that it knows where to go clean when it is placed in the pool. The engineers at Pentair imagine this as a cleaning frequency map of the pool, one that the cleaner can use to develop a path between dirty areas that ensures that all dirty areas of the pool are covered in the shortest amount of time with the least amount of overlap. However, the creation of this cleaning frequency map will take time as the cleaner will have to learn how often each area of the pool gets dirty. Thus, at the beginning of the cleaners operating cycle in a new pool there will be a learning period that could last days or weeks. During this learning period the ARPC will perform complete cleaning of the pool using the onboard SLAM system while monitoring how dirty each area of the pool is. Using the information that is collected over time the cleaner will generate the cleaning frequency map. Once the map has been generated the ARPC will use the cleaning frequency map to determine what locations of the pool should be cleaned each time it is placed in the water. However, during each cleaning operation the ARPC will continuously monitor how dirty each area of the pool is and update the cleaning frequency map. This continuous updating will allow the ARPC to learn how the amount of dirt located in each area of the pool might change throughout the year.

1.1.2. User Free Operation

The second key aspect of the ARPC is that designers at Pentair would like the ARPC to be truly autonomous, that is the cleaner can run itself with the least amount of user intervention possible. The goal is to develop the ARPC in such a way that it can enter the pool each time that it must clean the pool without the need of user assistance, go to the areas of the pool that are dirty and clean them, and return to its storage location. This truly autonomous design goal lead to several specific design choices in the development of the ARPC. First, in order to prevent the entanglement issue previously discussed with cleaners that receive their power from an external tether the ARPC was designed to be completely battery powered. By running off of battery power there is no need for an external power cord to be attached to the cleaner, thus lessening the chance of the cleaner getting stuck during operation. The second design choice that resulted is the use of a base station for the cleaner to *live* at. The base station will be located at the side of the pool and will store the ARPC when not in operation. When the time comes for the cleaner to clean, that is when the cleaner decides that there is a dirty area of the pool that must now be cleaned, the cleaner leaves the base station by itself and performs the required cleaning. Once the cleaning has been completed the ARPC returns to the base station to unload the captured debris and to charge the onboard batteries.

1.1.3. ARPC Prototype

With the overall goal of the ARPC developed, an undergraduate senior design team developed an initial prototype for the ARPC during the $2011-2012$ school year. The initial prototype, Figure 1.2, generated several unique design choices. First, a unique propulsion system was developed for the ARPC. Most current robotic pool cleaners are underwater wheeled vehicles that drive along the bottom of the pool. However, this drive mechanism

tends to be one of their primary failure points. The gears and belts wear out quickly and must replaced frequently. The ARPC uses a set of vibrating motors attached to angled brushes mounted to the bottom of the cleaner (Figure 1.2a). By using angled brushes a propulsive force is generated that moves the ARPC forward along the bottom of the pool. By turning off the vibrating motors on once side of the vehicle a turning motion can be achieved, thus the ARPC can be controlled in a method similar to a skid steered ground vehicle. To provide the suction to collect debris from the pool floor a set of pumping motors are attached to the vehicle (Figure 1.2b). These motors are attached to the outlet of the filter system, thus water is pulled through the suction inlet located on the bottom of the vehicle, through the onboard filter cartridge (Figure 1.2c) and expelled at the outlet where the motors are attached. Attached to the top of the ARPC is a skimming device (Figure 1.2d). This skimmer is used by the ARPC when it operates on the pool surface to force debris, such as leaves, into the filter cartridge while preventing the intake of large sticks that could clog the filter cartridge.

Following, the development of the initial prototype the design was taken to engineers at Pentair for evaluation. The engineers approved of the overall form of the initial prototype however, they took issue with the overall size and weight of the cleaner. The initial prototype was constructed primarily of off the shelf components which resulted in the ARPC being extremely large and bulky. With the overall design approved a Virginia Tech masters student spent the Fall of 2012 optimizing the design. This included the replacement of many of the bulky off the shelf components with custom designed and 3D printed components. While all of the unique features of the initial prototype remained the size and weight was reduced significantly. The updated ARPC prototype is shown in Figure 1.3 with the corresponding unique design features labeled.

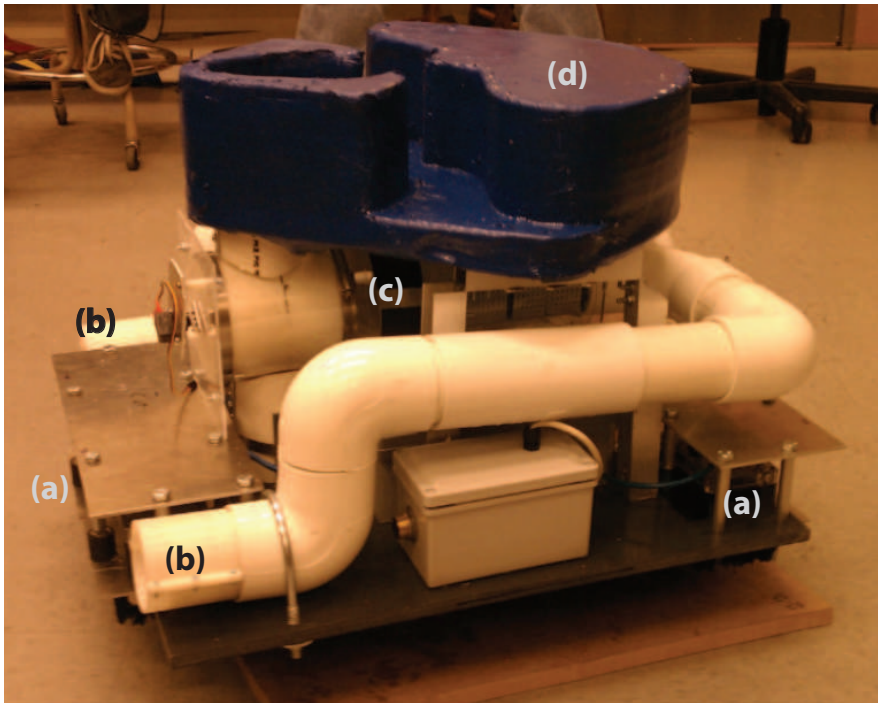


Figure 1.2: Initial ARPC prototype developed by VT senior design team.

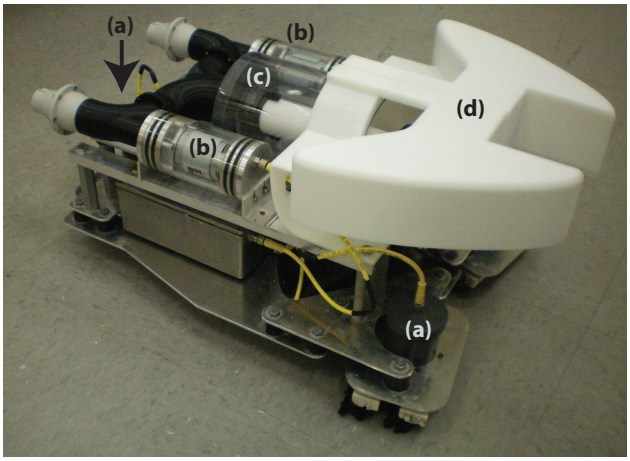


Figure 1.3: Second ARPC prototype with the vibrating propulsion motors (a), suction pumps (b), filter cartridge (c), and skimming device (d).

1.2. Contribution

The goal of the research described in this dissertation is to develop a system that allows for a small unmanned underwater vehicle (UUV), similar to the one described in Section 1.1.3, to solve the SLAM problem using inexpensive sensors and low cost computing hardware. As a results the contributions of this dissertation come in two forms: i) the selection, or development, of a suite of low cost sensors that can provide the UUV with the information required to solve the SLAM problem and ii) the development of a software algorithm that allows he UUV to solve the SLAM problem in real-time using the low cost computing resources available onboard.

1.2.1. Hardware

In order for the UUV to be able to solve the SLAM problem it must be equipped with sensors that provide localization information, typically in the form of odometry information, and perceptive measurements about objects that surround it in the environment. The first significant contribution is found in Chapter 3 where the design of a prototype underwater rangefinder is presented. The rangefinder uses two laser line generators and a camera to calculate the distance to obstacles in an underwater environment.

1.2.2. Software

The remaining significant contributions of this dissertation are related to the development of a real-time SLAM algorithm that can be performed using low-cost and memory challenged computing resources. In Chapter 8 a SLAM algorithm is presented that stores the occupancy grid representing the environment using an alternative basis, specifically the Haar Wavelet basis. The last significant contribution of this dissertation is presented in Chapter 9 where a solution to the SLAM problem that makes use of compressed occupancy grids is developed.

1.3. Outline

The remainder of this dissertation is organized as follows. In Chapter 2 preliminary materials are presented including a review of relevant literature related to the research presented in this dissertation and a series of relevant mathematical preliminaries. In Chapter 3 the design of our underwater rangefinder is presented along with experimental results that illustrate the performance of the sensor. In Chapter 4 a visual odometry approach using a downward facing camera is presented and the performance of the approach is examined using experimental results. In Chapter 5 and Chapter 6 the suite of underwater sensors is validated using two common feature based solutions to the SLAM problem, the approach that makes use of an extended Kalman filter (EKF SLAM) and the approach that solves the problem using a Rao-Blackwellized particle filter (FastSLAM). In Chapter 7 an extension to the standard FastSLAM algorithm is presented that stores the map of the environment using occupancy grids. In Chapter 8 a solution to the SLAM problem that extends that described in Chapter 7 is presented that stores the occupancy grid using the Haar wavelet basis representation. In Chapter 9 a solution to the SLAM problem that stores a compressed form of the occupancy grid is presented. Finally, concluding remarks and areas of future research are presented in Chapter 10.

Chapter 2

Preliminary Material

2.1. Literature Review

2.1.1. Laser Range Finder

The use of cameras and lasers for distance calculations and mapping have become an area of interest recently. One of the primary reasons for this interest is their low cost. For a few hundred dollars a sensor can be designed to provide measurements similar to those provided by much more expensive systems. A second advantage of using lasers and cameras is that when placed inside of a waterproof enclosure they can operate in underwater environments where many other typically used sensors cannot operate. In [4] the authors develop a distance measurement sensor based around the use of a single laser pointer mounted on an underwater vehicle. A similar system is developed in [5] except the authors use a pair of laser pointers which are used to calculate the distance to an object along with the orientation. In [6] the authors developed a system that uses a laser line generator to build a three dimensional profile of the seafloor. In [7] a similar system is developed, however the floor profile generated by the laser and camera system is used to determine a suitable location for landing an underwater vehicle. Finally, the authors of [8, 9, 10] design a sensor based around the use of a laser pointer and mirror. The system is used to generate three dimensional maps of enclosed

areas, with their final goal being the development of a sensor that can be attached to robots operating in areas of heavy rubble so that three dimensional maps of the enclosed areas inside of the rubble can be generated.

2.1.2. Visual Odometry

To aid in vehicle localization one of the most commonly used sensor is an odometry sensor. Odometry sensors provide change in position information for the vehicle. In many cases the necessary measurements are provided by encoders mounted to the drive motor, or directly to the wheel shafts of the vehicle, and they count the number of revolutions the motor or wheels make. However, in some cases encoders cannot be used, either due to the form factor of the vehicle or the environment in which the vehicle operates. In these cases a sensor that is used to provide odometry information is a downward facing camera. In [11] a downward facing camera is mounted to a robotic vehicle that has Mecanum wheels attached, which makes using wheel encoders difficult. The visual odometry system tracks multiple points of interest (feature points) between image frames to determine the translational and rotational change in the vehicles pose. The system is then tested using several different control schemes and the corresponding results are compared to those using wheel encoders. In [12] a downward facing camera is mounted to a differential drive vehicle. The cross-correlation between two images, that have been passed through a Fast Fourier Transform (FFT), is used to determine the translational and rotational changes in position of the vehicle. The authors of [13] use two cameras, one downward facing and one angled down but forward looking, to provide odometry data. Optical flow techniques are used to calculate the odometry data and a Kalman filter fuses the data from the two cameras. The authors of [14] attached a downward facing camera to a passenger automobile. A correlation based template matching approach is used to calculate the translational changes between

frames. However, due to the high speeds of the vehicle a restricted search region is used to improve processing times. The search region is determined based on previous odometry measurements. Also, to increase the robustness of the system a template quality score is developed which describes the uniqueness of a template region. Using this score the authors select from several different possible templates one that they believe will lead to the best possible correlation result. Finally, in [15] a odometry system is designed using off the shelf optical flow sensors which are originally designed for use in computer mice. The developed system is intended to be used on a planetary rover vehicle.

The algorithms used to calculate visual odometry data using a downward facing camera tend to fall into two categories. A family of algorithms referred to as *sparse* algorithms generate odometry information by analyzing a subset of the pixels in the full image frame. A common example of a sparse algorithm is template matching. In template matching the location of a small subsection of one image is located in a second image and in visual odometry applications the change in position of the template provides the change in position of the camera that acquired the image. In many cases this template matching in performed through correlation, as an example the system described in [14]. In order to improve the performance of a system of this type a new approach to template matching makes use of the wavelet decomposition of the image being searched and the template, [16] and [17] are two examples. By transforming both of the images using the wavelet transform the template matching can be performed at multiple resolutions. The general area in which a template image is located is first found at a low resolution, then the specific location is found in a higher resolution version of the image. By performing the template matching a different resolutions the computing time is reduced because when the general location of the template is being found, the correlation is performed on less data than in the case when the entire image is searched at full resolution.

The second family of algorithms used to calculate visual odometry data are referred to as *dense* algorithms as they attempt to calculate a velocity vector for every pixel in the acquired image. Two common dense algorithms are the Horn-Schunck method ([18]) and the Lucas-Kanade algorithm ([19]). An area of research related to these dense algorithms is particle image velocimetry (PIV), [20] provides a nice overview of the history and current status of PIV research. In PIV algorithms similar to dense visual odometry algorithms are used to track particles in a video stream. PIV is typically used in the study of fluid flows in which a sheet laser a projected through the fluid being examined and video of the flow is acquired. The video is then processed to track particles in the flow in order to gain a better understanding of what is happening in different regions of the flow.

2.1.3. Gaussian SLAM

The first, and possibly most common, solution to the Simultaneous Localization and Mapping (SLAM) problem came as a result of several important papers [21, 22] and it solves the SLAM problem using an Extended Kalman Filter (EKF). In the most common implementation of this solution, the environment in which the robot operates is represented by a set of discrete locations that can be easily extracted from sensor data, referred to as *landmarks*. The EKF solution assumes that the noise on the sensors is additive and Gaussian. The result is that the posterior representing the full SLAM state, composed of a vehicle pose and map, is Gaussian, thus it can be represented by a mean vector and corresponding covariance matrix. The EKF linearizes any non-linearities in the system models using a first order Taylor approximation.

While the EKF solution is the most common solution to the SLAM problem representing the state as a Gaussian distribution there are several other solutions that have been proposed. One alternate solution to the SLAM problem uses an Unscented Kalman Filter

(UKF) to estimate the mean and covariance ([23]). The UKF performs the linearization of the models by using a set of samples to estimate the first three moments of the prior distribution. Another alternate solution to the Gaussian SLAM problem is referred to as GraphSLAM ([24]) which uses the sensor data to generate a set of constraints for the graph and then maximizes the set of constraints to generate a maximum likelihood map. The GraphSLAM approach differs from the EKF and UKF solutions in that it estimates the full vehicle trajectory, not just the instantaneous pose, along with the map.

There are issues with each of the solutions discussed so far. The EKF and UKF solutions to the SLAM problem estimate the instantaneous pose of the robot along with the map. However, the update complexity of these solutions increases quadratically with the number of landmarks used to represent the map. The GraphSLAM algorithm is more efficient, however it accumulates information and uses all of the information to solve the SLAM problem after the fact. An approach has been developed that attempts to incorporate the benefits of these two approaches to solve the online, or real time, SLAM problem. The combined approach is referred to as Sparse Extended Information Filters ([25]). The Sparse Extended Information Filter solution integrates out past pose information, similar to the EKF, while maintaining all of the previous knowledge using an information filter in order to estimate the SLAM state in real time.

The literature contains a large number of published examples of real-world implementations of Gaussian SLAM solutions using a wide variety of robotic vehicles operating in many different environments. In the remainder of this section we will introduce several implementations for a wide range of operating environments. In [26] the authors use a single camera mounted to a ground vehicle and the EKF solution. Due to the fact that a single camera is being used distance information can not be inferred about objects observed by the camera, only the bearing to the object can be calculated. In order to overcome the lack of distance

information a feature initialization process is developed that involves observing a landmark from separate robot poses. In [27] the 3D SLAM problem in indoor environments is solved using the EKF SLAM approach. A single laser rangefinder is rotated to generate 3D point clouds for feature extraction and mapping. The authors of [28] mounted a single camera and laser rangefinder to a mobile ground vehicle operating in an indoor environment and use the fused data to solve the SLAM problem using the EKF approach. In [29, 30] an inertial measurement unit (IMU) and single camera are mounted on an unmanned aerial vehicle. The mounted sensors are used along with the EKF SLAM solution to localize the vehicle and map artificial visual features that are located on the ground.

2.1.4. Non-Parametric SLAM

Along with the family of solutions that estimate the SLAM state as a Gaussian distribution, a second family of solutions exists which do not rely on the Gaussian assumption. This family of solutions is based upon particle filters which are estimators that sample multiple possible states based on one set of measurements and select the best possible state that matches a second set of measurements. The use of particle filters in robotics applications was introduced in [31] for localization purposes. Particle filters were first introduced in the context of SLAM in [32]. A very popular solution to the SLAM problem using particles filters is referred to as FastSLAM and introduced in [33]. The FastSLAM algorithm uses a Rao-Blackwellized particle filter to estimate the pose of the robot while each particle uses a set of EKFs to estimate the location of landmarks in the environment. An update of the FastSLAM algorithm is presented in [34]. The difference between the two FastSLAM approaches is in how robot poses are sampled. In FastSLAM 1.0 ([35]) the poses are sampled based solely on the control inputs measured by the robot however, FastSLAM 2.0 ([34]) includes the measurements in the pose sampling step. One of the advantages of the particle

filter based solution to the SLAM problem is that the different types of maps can be maintained by each particle. The use of occupancy grids in particle filter based SLAM solutions was introduced in [36]. However, one of the primary issues with the use of occupancy grids with particle filters is the amount of memory required. In [37] an approach was developed for representing occupancy grid maps for a set of particles. A tree structure is used to store the updates made to the map by each particle as opposed to storing a unique map for every single particle.

2.1.5. Feature Extraction

For SLAM algorithms that represent the world as a set of landmarks one of the largest areas of research regards how to extract features from sensor data. In [38] a feature extraction method is developed for inexpensive sonar sensors. The system designed by the authors extracts both point features and line features, and experimental results are provided using the extraction method in a SLAM framework. In [39] a method is developed to detect outliers in LiDAR point clouds generated by an airborne LiDAR. The outlier detector uses kd-trees, a k-dimensional tree which is a data structure used for organizing points in a k-dimensional space, to detect and exclude the outliers. The authors of [40] evaluate the effectiveness of six separate line detection algorithms on laser range finder data. In [41] the authors develop an algorithm that can extract several types of features from LiDAR data: i) lines, ii) arcs or circles, and iii) people's legs. The developed algorithm was implemented as a plugin to the Player ([42]) robotics simulation platform. Finally, the authors of [43] develop a general purpose feature extractor for extracting features from LiDAR data in any environment.

2.1.6. Underwater SLAM

One environment in which SLAM implementations are less common, as opposed to ground vehicles operating indoors and outdoors or aerial vehicles, is underwater environments. While the underwater environment is less common, there are still several notable implementations in the literature that were examined. In [44] a feature based SLAM method is developed for underwater environments. The system uses the Robust Extended Kalman Filter as opposed to the classic EKF. The authors of [45] developed a system for mapping and localizing a small exploratory Remotely Operated Vehicle (ROV) that operates in underground cistern systems. The ROV was equipped with a depth sensor, compass, and scanning sonar. A dynamic vehicle model is used to implement a FastSLAM algorithm using occupancy grids. The SLAM algorithm was not used in real-time, rather the vehicle was remotely controlled by a user and the collected data was used to determine the path that the vehicle travelled along with a map of the cistern environment after the run. An underwater SLAM approach is developed in [46] and based around the use of computer vision. The vehicle uses attached stereo cameras and a visual odometry system is used to estimate the motion of the vehicle. The authors implement the EKF and UKF solution to the SLAM problem. However, a separate smoothing step is included to smooth all of the measurements. A Rauch-Tung-Striebel Smoother is to smooth the entire control and measurement histories. A SLAM system that uses low frequency high resolution images is developed in [47]. The system makes use of on board inertial sensors and a high resolution underwater camera to implement a SLAM approach based on the sparse extended information filter. The system is used to localize a ROV over a 3km track while mapping the RMS Titanic.

2.1.7. Sparse Signal Recovery

Research into sparse signal recovery is of great importance as it serves as the basis for data compression. One of the current areas of interest in the research of sparse signal recovery is referred to as compressed sensing ([48]). Compressed sensing provides a framework in which a sparse signal can be recovered accurately with fewer measurements than would be required according to the Nyquist/Shannon sampling theorem ([49]). In [50] the authors improve on the standard compressed sensing approach using the knowledge that in many situations the non-zero elements in a sparse signal have a particular structure that can be exploited. They use graphical models to exploit the structure of the non-zero elements in a sparse signal in order to improve on the standard compressed sensing approach. In [51] the authors exploit the fact that in many cases when a signal is decomposed using some multiscale basis the non-zero elements in the resulting signal are organized into *trees*. The authors use this tree structure as prior knowledge to reconstruct the sparse signal from a small number of measurements as a linear inverse problem. The authors of [52] use information theory to derive the necessary conditions for the exact reconstruction of a sparse signal from a small number of measurements when a dense measurement matrix is used, such a Gaussian matrix. Secondly, they develop a lower bound on the number of measurements required to reconstruct a sparse signal when the measurement matrix is sparse, that is each row of the measurement matrix is composed primarily of zeros.

2.2. Mathematical Preliminaries

2.2.1. Notation

There are three standard data types that are used throughout this dissertation. Scalar values are denoted by italic values, for example $x \in \mathbb{R}$ denotes a scalar value that is a member

of the set of real number $\mathbb{R}$. Vectors are identified using bold lower case notation, for example $\mathbf{x} \in \mathbb{R}^n$ is a vector containing $n \in \mathbb{N}$ elements where $\mathbb{N}$ is the set of natural numbers. The lth element in a vector is referenced as $\mathbf{x}(l)$. Matrices are displayed using bold uppercase notation, for example $\mathbf{X} \in \mathbb{R}^{h \times w}$ is a matrix with $h \in \mathbb{N}$ rows and $w \in \mathbb{N}$ columns. A single element in the matrix is referenced according to $\mathbf{X}(i, j)$ which corresponds to the element in $\mathbf{X}$ located in the ith row and jth column. In many cases it is desirable to reference a single row or column in a matrix, in this case a ":" is placed in the second dimension, for example the ith row of $\mathbf{X}$ is referenced as $\mathbf{X}(i, :)$ and the jth column is referenced as $\mathbf{X}(:, j)$.

2.2.2. Probability Concepts

In much of the literature, and as a result in much of this dissertation, the SLAM problem is addressed from a probabilistic point of view. Due to this fact several introductory concepts in probability theory are introduced in this section to familiarize the reader. The form of the presented probability theory is taken from [53] but the same concepts can be found in most introduction to probability theory textbooks. In many cases we are attempting to estimate the probability density function of some random variable $x \in \mathbb{R}$. The *probability density function* that describes the random variable x is denoted

$$p(x), \tag{2.1}$$

and is often just referred to as the *distribution* of x. In many situations we attempt to estimate the distribution that describes two random variables, $x \in \mathbb{R}$ and $y \in \mathbb{R}$. The *joint distribution* that describes the distribution of x and y is given as

$$p(x, y). \tag{2.2}$$

If x and y are independent then (2.2) can be rewritten as a product of the two distributions

$$p(x, y) = p(x)\, p(y). \tag{2.3}$$

In many cases the random variable $y \in \mathbb{R}$ provides some information about the random variable $x \in \mathbb{R}$, in this case the *conditional probability* is written as

$$p(x \mid y). \tag{2.4}$$

If $p(y) > 0$ then (2.4) can be written as

$$p(x \mid y) = \frac{p(x, y)}{p(y)}, \tag{2.5}$$

and if x and y are independent then (2.4) can be written as

$$p(x \mid y) = \frac{p(x)\, p(y)}{p(y)} = p(x). \tag{2.6}$$

A key relationship that is used throughout this dissertation is built on the idea of conditional independence and denoted the *Theorem of Total Probability*. The Theory of Total Probability for a random variable $x \in \mathbb{R}$ is given as

$$p(x) = \int p(x \mid y)\, p(y)\, dy. \tag{2.7}$$

A second important relationship used throughout this dissertation is referred to as the *Bayes Rule* and it relates a conditional probability to its inverse according to

$$p(x \mid y) = \frac{p(y \mid x)\, p(x)}{p(y)} = \frac{p(y \mid x)\, p(x)}{\int p(y \mid x')\, p(x')\, dx'}. \tag{2.8}$$

Examining (2.8) it can be seen that the denominator $p(y)$ is not dependent on x so it is common to treat the denominator as a normalizing factor. In this scenario (2.8) is rewritten as

$$p(x \mid y) = \eta p(y \mid x)\, p(x). \tag{2.9}$$

Finally, in many cases when dealing with the distribution of $x \in \mathbb{R}$ that is conditional on $y \in \mathbb{R}$ we refer to the distribution of x before the inclusion of y, $p(x)$ as the *prior distribution* which is often shortened to *prior*. With the inclusion of y the conditional probability, $p(x \mid y)$ is referred to as the *posterior distribution* which is often shortened to *posterior*.

2.2.3. The SLAM Problem as a Recursive Bayesian Estimation Problem

A key insight that has affected the form taken by many solutions to the SLAM problem is that the SLAM problem can be viewed as a recursive Bayesian estimation problem. In this section we will follow the derivation from [54] to show that the SLAM problem can be treated as a Bayesian estimation problem as this idea serves as the basis for all of the solutions discussed in this dissertation. In a probabilistic sense in solving the SLAM problem we are attempting to estimate the distribution

$$p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{2.10}$$

that represents the pose of an unmanned vehicle (UMV), $\boldsymbol{\xi}_k$, at time step k and the map of the environment in which the UMV operates, $\mathbf{M}$, based on a series of control inputs $\mathbf{u}_{1:k}$ and observations $\mathbf{z}_{1:k}$. In this dissertation we treat the world in which the UMV operates as a two dimensional plane, thus $\boldsymbol{\xi}_k \triangleq [\ x_k \quad y_k \quad \theta_k\ ]^{\mathrm{T}}$ where $[\ x_k \quad y_k\ ] \in \mathbb{R}^2$ are the horizontal and vertical position of the vehicle in some frame of reference and $\theta_k \in (-\pi, \pi]$ is the heading of the UMV with respect to the positive horizontal axis of the frame of reference. For this derivation it is assumed that the environment in which the UMV operates is static so the time step index is dropped from the map. However, in later chapters when the map is being incrementally constructed the time step index will be included when appropriate. The series of sensor measurements $\mathbf{z}_{1:k}$ denotes all of the sensor measurements for $k > 0$, thus $\mathbf{z}_{1:k} \triangleq [\mathbf{z}_1, \ldots, \mathbf{z}_k]$ where $\mathbf{z}_i \in \mathbb{R}^m$, $i = 1, 2, \ldots, k$ and $m \in \mathbb{N}$ is the number of measurements in $\mathbf{z}_i$. In the same way the full series of control inputs are defined as $\mathbf{u}_{1:k} \triangleq [\mathbf{u}_1, \ldots, \mathbf{u}_k]$ where $\mathbf{u}_i \in \mathbb{R}^c$, $i = 1, 2, \ldots, k$ and $c \in \mathbb{N}$ is the size of the control vector $\mathbf{u}_i$.

Using (2.9), the joint distribution that we are attempting to estimate (2.10) can be

rewritten as

$$p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right), \tag{2.11}$$

where $\eta > 0$ is a normalization factor. We now exploit the idea of independence in noting that the current measurement $\mathbf{z}_k$ is only dependent on the current pose, $\boldsymbol{\xi}_k$, and map, $\mathbf{M}$, thus (2.11) is rewritten as

$$p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right). \tag{2.12}$$

Using (2.7) the rightmost term in (2.12) is expanded as

$$\begin{aligned} p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) =& \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) \int p\left(\boldsymbol{\xi}_k, \mathbf{M} \;\middle|\; \boldsymbol{\xi}_{k-1}, \mathbf{u}_{1:k}\mathbf{z}_{1:k-1}\right) \\ & \times p\left(\boldsymbol{\xi}_{k-1} \;\middle|\; \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) d\boldsymbol{\xi}_{k-1}. \end{aligned} \tag{2.13}$$

Using the concept of conditional probability the leftmost term under the integral in (2.13) can be expanded to yield

$$\begin{aligned} p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) =& \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) \int p\left(\boldsymbol{\xi}_k \;\middle|\; \mathbf{M}, \boldsymbol{\xi}_{k-1}, \mathbf{u}_{1:k}\mathbf{z}_{1:k-1}\right) \\ & \times p\left(\mathbf{M} \;\middle|\; \boldsymbol{\xi}_{k-1}, \mathbf{u}_{1:k}\mathbf{z}_{1:k-1}\right) p\left(\boldsymbol{\xi}_{k-1} \;\middle|\; \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) d\boldsymbol{\xi}_{k-1}. \end{aligned} \tag{2.14}$$

Using the notion that $\boldsymbol{\xi}_k$ is only a dependent on $\boldsymbol{\xi}_{k-1}$ and $\mathbf{u}_k$ (2.14) is simplified to

$$\begin{aligned} p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) =& \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) \int p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}, \mathbf{u}_k\right) \\ & \times p\left(\mathbf{M} \;\middle|\; \boldsymbol{\xi}_{k-1}, \mathbf{u}_{1:k}\mathbf{z}_{1:k-1}\right) p\left(\boldsymbol{\xi}_{k-1} \;\middle|\; \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) d\boldsymbol{\xi}_{k-1}. \end{aligned} \tag{2.15}$$

Now the right most terms under the integral in (2.15) can be combined to yield

$$\begin{aligned} p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) =& \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) \int p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}, \mathbf{u}_k\right) \\ & \times p\left(\boldsymbol{\xi}_{k-1}, \mathbf{M} \;\middle|\; \mathbf{u}_{1:k}\mathbf{z}_{1:k-1}\right) d\boldsymbol{\xi}_{k-1}. \end{aligned} \tag{2.16}$$

Finally, as $\mathbf{u}_k$ provides no information about the pose at time step $k-1$ the final form of the SLAM posterior is given as

$$p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = \eta p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right) \int p\left(\boldsymbol{\xi}_k \mid \boldsymbol{\xi}_{k-1}, \mathbf{u}_k\right) \times p\left(\boldsymbol{\xi}_{k-1}, \mathbf{M} \mid \mathbf{u}_{1:k-1}\mathbf{z}_{1:k-1}\right) d\boldsymbol{\xi}_{k-1}. \tag{2.17}$$

This form of the SLAM posterior proves to be quite useful. It allows for the SLAM posterior, (2.10), to be estimated using any number of recursive Bayesian estimation techniques. The typical recursive Bayesian estimator estimates the SLAM posterior using a *predictor-corrector* approach. The prediction component of the estimator implements the portion of (2.17) under the integral, which generates a new pose estimate using the probabilistic motion model of the UMV, $p\left(\boldsymbol{\xi}_k \mid \boldsymbol{\xi}_{k-1}, \mathbf{u}_k\right)$, the pose estimate from the previous time step, $\boldsymbol{\xi}_{k-1}$, and the current control input $\mathbf{u}_k$. The correction takes place by incorporating the current observation measurement into the estimate through the probabilistic measurement model of the system, $p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k, \mathbf{M}\right)$. More information on this will be provided in later chapters.

Chapter 3

Camera and Laser Rangefinder

3.1. Introduction

When considering unmanned vehicle applications, the ability to locate objects and be aware of the surroundings is crucial. This information is needed for high level applications such as mapping and vehicle localization, as well as low level applications such as obstacle avoidance. Light Detection and Ranging (LiDAR) sensors do an extremely good job of providing a vehicle with information about the surrounding environment however, they have a high cost that is prohibitive for low budget applications and commercially available sensors are unable to operate in underwater environments. Sound Navigation and Ranging (SONAR) sensors work well determining distances in underwater environments, however in enclosed environments SONARs are difficult to use due to the high number of multiple returns caused by reflections.

A prototype for a low cost distance measuring sensor using a single camera and parallel laser line generators has been developed. Similar sensors have been proposed although our sensor presents a few main differences with respect to those described in the literature. In [4, 5] sensors using a single laser pointer projecting a dot are considered. Because of their design, they are only able to calculate distance information related to a single location

directly in front of the camera. Also, both of these designs rely heavily on calibration routines that map the pointer's location in the image frame with a distance. We will demonstrate how our approach removes the need for this calibration step. Other, similar approaches have been developed for different tasks. In [6, 7] the use of a single laser line is used to generate 3D texture maps of underwater objects.

The remainder of this chapter is organized as follows. Section 3.2 discusses the physical layout of the sensor. Section 3.3 describes the pinhole camera model on which the distance calculation is based. Section 3.4 discusses the theory behind the distance calculation. Section 3.5 discusses all of the steps of the image processing algorithm used to extract the laser line locations from the image. Experimental results are provided in Section 3.6. An error analysis is performed in Section 3.7 and concluding remarks are presented in Section 3.8.

3.2. Physical Layout

The physical design of the sensor comes from the structured light approach that serves as the basis for the sensor. The sensor's physical layout can be seen in Figure 3.1 and is composed of a pair of laser line generators (A and B) and a CCD camera (C). The laser line generators are mounted so that the laser lines are parallel to each other and orthogonal to the viewing axis of the camera. The result is that two parallel laser lines are projected horizontally across the image captured by the camera. The camera selected for the prototype is a Sony FCB-EX11D ([55]) which uses a 1/4-type CCD sensor, is equipped with a 10x optical zoom and can provide a 120x magnification when combined with the camera's digital zoom.

The laser line generators are Apinex GM-CW02L ([56]) which produce green lines with a wavelength of 532nm, have a power rating < 50mW, and a produce lines across a 90° fan angle. The 532nm wavelength was chosen because it has a low absorption coefficient

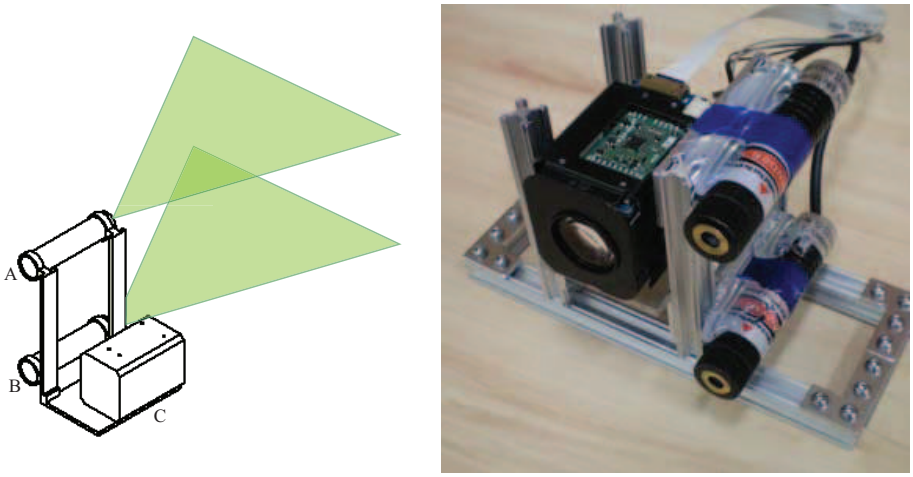


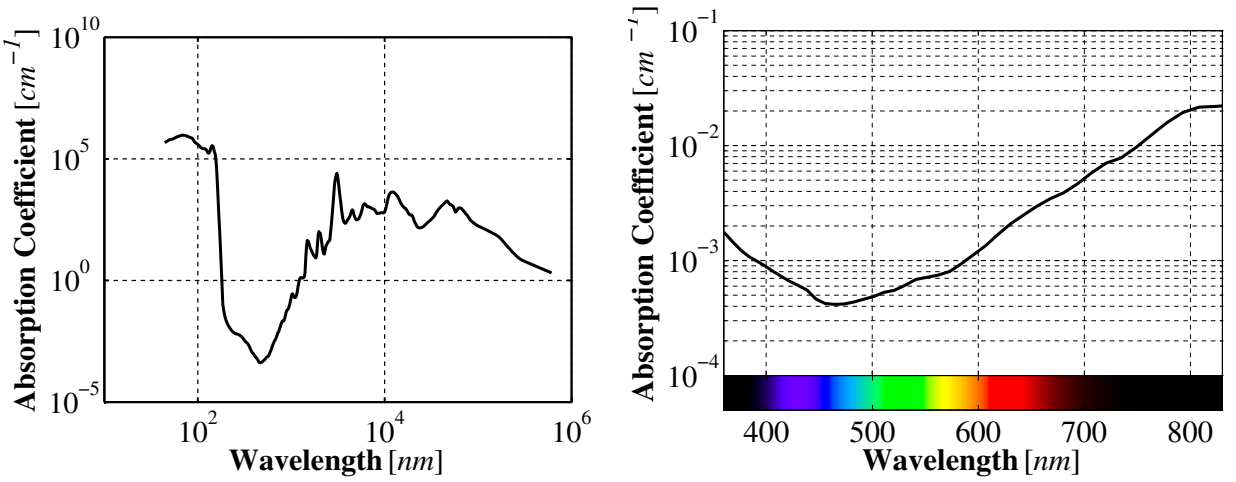
Figure 3.1: Design of the prototype sensor on the left and the actual prototype on the right.

in water (Figure 3.2). Other colors have lower absorption coefficients, primarily blue and violet, however at the time the prototype was developed laser line generators producing those colors were found to be much more expensive than those that produced light with the green wavelength.

3.3. Pinhole Camera Model

The method used by our sensor to measure the distance to an object is based on the pinhole camera model ([57]). The pinhole camera model relates a point in the world to its projection on the camera's focal plane. According to the pinhole camera model the light reflected off an object located in the world at $\mathbf{p} = (x, y, z)$ that passes through the camera's aperture located at $\mathbf{o} = (0, 0, 0)$ is projected onto the focal plane of the camera at $\mathbf{q} = (u, v, -f)$; this relationship is illustrated in Figure 3.3. By examining the projection in the xz plane (Figure 3.4a) and the yz plane (Figure 3.4b) the relationship between $\mathbf{p}$ and $\mathbf{q}$ is given by

$$\frac{x}{z} = -\frac{u}{f} \text{ and } \frac{y}{z} = -\frac{v}{f}, \tag{3.1}$$



(a) (b)

Figure 3.2: Absorption coefficient of light in water as a function of wavelength over the total spectrum on the left and the visible spectrum on the right (obtained using data from [1]).

where $(x, y, z) \in \mathbb{R}^3$ are the components of $\mathbf{p}$ in the real world, $f \in \mathbb{R}$ is the focal distance of the camera being used, and $(u, v) \in \mathbb{N}^2$ are the horizontal and vertical components of $\mathbf{q}$ measured in the camera's unit of measure pixels. The negative sign in (3.1) comes from the fact that light passing through the camera's aperture is mirrored across the horizontal and vertical axis onto the camera's focal plane located behind the camera's aperture as shown in Figure 3.3.

Equation (3.1) is simplified by placing the camera's focal plane in front of the aperture. Using the simplified pinhole camera model, whose projection in the xz plane is seen in Figure 3.5a and in the yz plane in Figure 3.5b, the relationship between $\mathbf{p}$ and $\mathbf{q}$ can be rewritten as

$$\frac{x}{z} = \frac{u}{f} \text{ and } \frac{y}{z} = \frac{v}{f}. \tag{3.2}$$

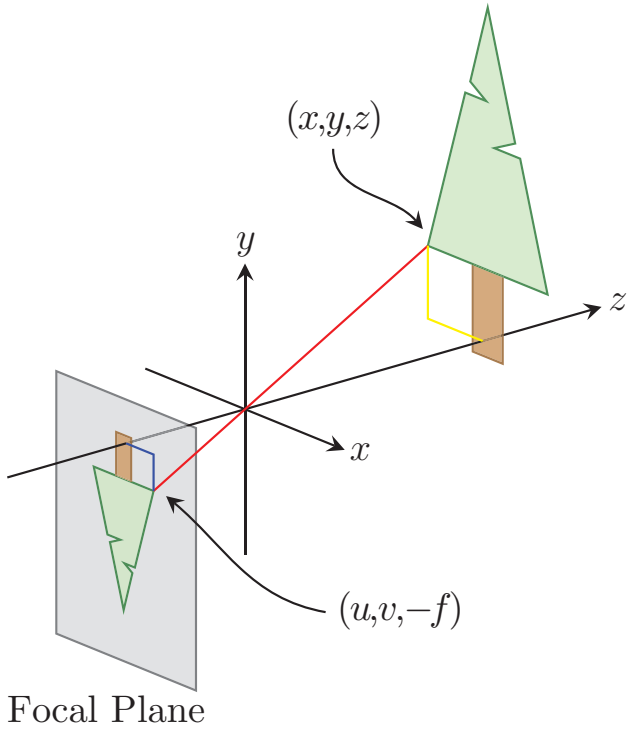


Figure 3.3: The classic pinhole model of a camera.

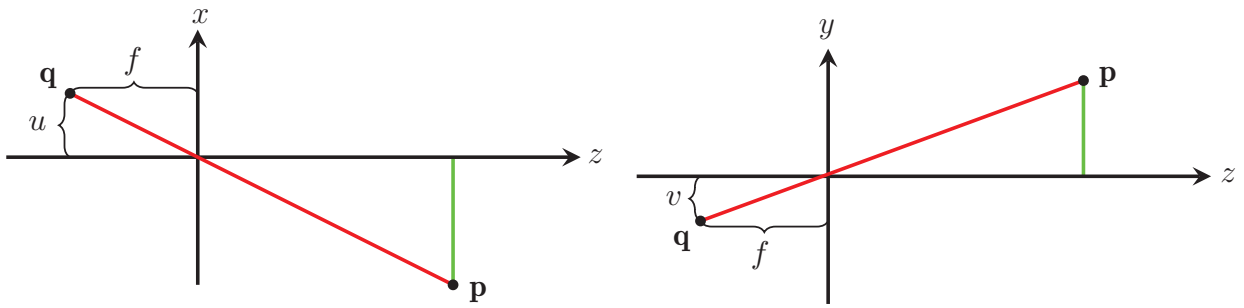


Figure 3.4: Projections of the classic pinhole camera model in the xz plane (a) and the yz plane (b).

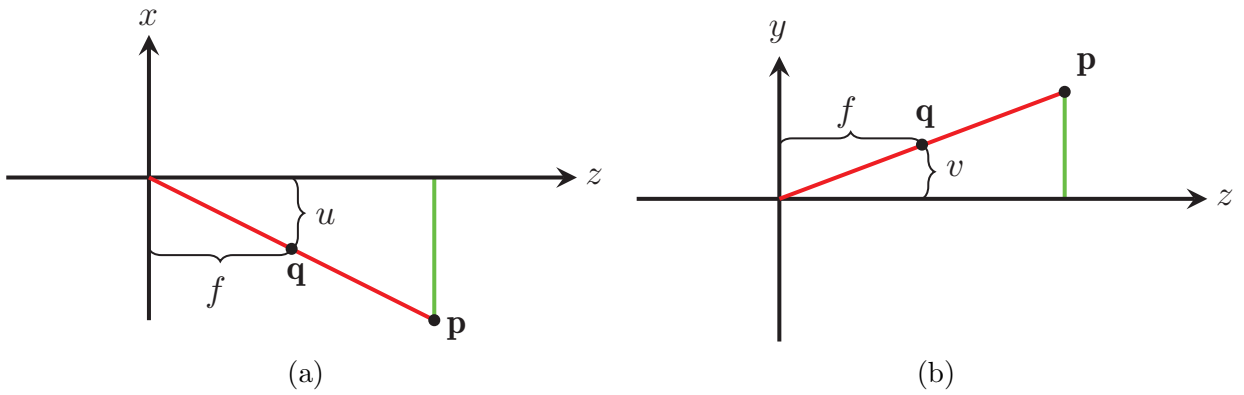


Figure 3.5: Projections of the modified pinhole camera model in the xz plane (a) and the yz plane (b).

3.4. Distance Measurement Theory

Our distance measurement is based on the physical layout of the sensor, whose side view is shown in Figure 3.6, and (3.2). In examining Figure 3.6 it can be seen that a pair of similar triangles is created between the camera's aperture and i) the projection of the laser lines on the camera's focal plane (**oab**) and ii) the location of the laser lines on an object (**ocd**) . By equating the two triangles, the relationship between the laser lines in world coordinates and their projection on the camera's focal plane is given by

$$\frac{\tilde{y}}{z} = \frac{\tilde{v}}{f}, \tag{3.3}$$

where $\tilde{y} \triangleq y_1 - y_2$ is the physical distance that separates the laser lines, $\tilde{v} \triangleq v_1 - v_2$ is the distance between the laser lines on the camera's focal plane, f is the focal length of the camera, and z is the unknown distance to the object.

The physical distance separating the laser lines, $\tilde{y}$, is known from the physical setup of the sensor and f can be found for the camera being used, hence only $\tilde{v}$ is needed to compute z. The distance that separates the laser lines in the image captured by the camera, $\tilde{v}$, is found through an image processing algorithm, described in Section 3.5. After $\tilde{v}$ has been

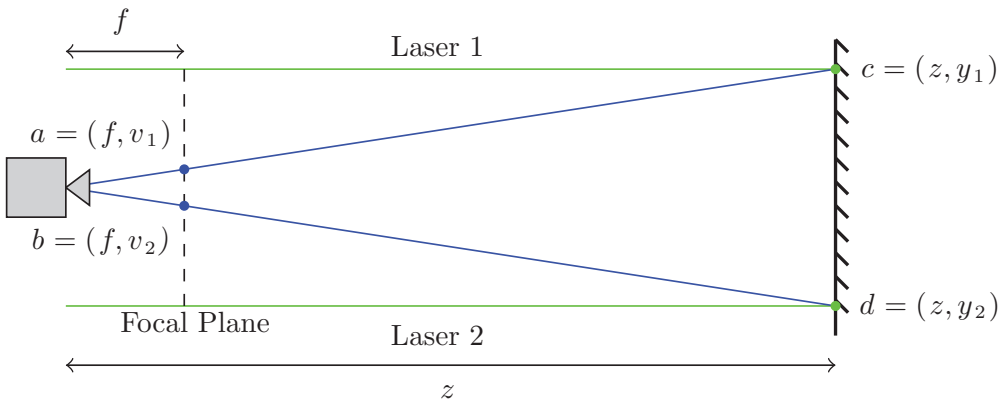


Figure 3.6: Side view of the sensor

found the distance to the object is calculated as

$$z = \tilde{y}\left(\frac{f}{\tilde{v}}\right). \tag{3.4}$$

3.5. Image Processing Algorithm

As seen in (3.4), to determine how far away an object is from the sensor it is necessary to determine the distance between the two laser lines in the image captured by the camera. To accomplish this, an algorithm was developed to extract the laser lines from an image. An overview of the algorithm is seen in Figure 3.7.

3.5.1. Distortion Removal

The first step of the image processing algorithm removes distortions that are present in the image due to lens and manufacturing defects. These distortions prevent the acquired image from behaving as expected based on the pinhole camera model so they must be corrected before the distance calculation can be made. The distortion model that was selected ([58]) assumes two types of distortion, radial and tangential. The relationship between a pixel location in the image and the expected location if the camera behaved according to the

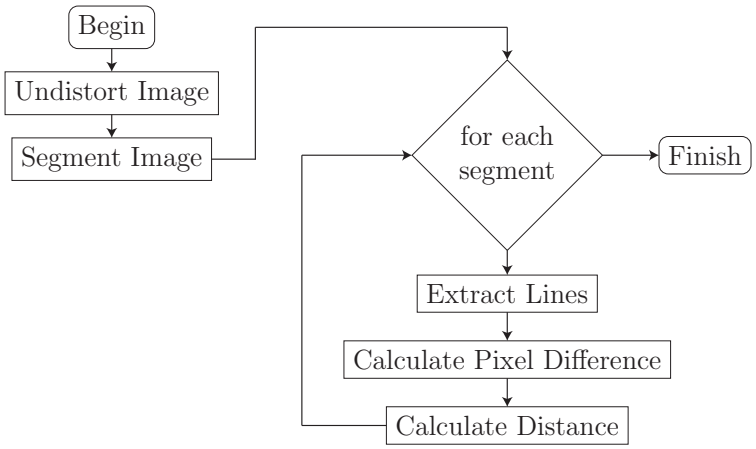


Figure 3.7: Overview of the distance calculation algorithm.

pinhole camera model is given by

$$u' = u + \tilde{u}\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right) + \left(p_1\left(r^2 + 2\tilde{u}^2\right) + 2p_2\tilde{u}\tilde{v}\right), \tag{3.5}$$

$$v' = v + \tilde{v}\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right) + \left(p_1\left(r^2 + 2\tilde{v}^2\right) + 2p_2\tilde{u}\tilde{v}\right), \tag{3.6}$$

where $(u', v') \in \mathbb{N}^2$ is where $(u, v) \in \mathbb{N}^2$ would be located if the camera behaved according to the pinhole camera model and u and v are the horizontal and vertical components of the pixel location in the image. The parameters $k_i \in \mathbb{R}$, $i = 1, 2, 3$ are the coefficients that correspond to the radial distortion and $p_j \in \mathbb{R}$, $j = 1, 2$ are the coefficients that describe the tangential distortion. The variables $u_c \in \mathbb{N}$ and $v_c \in \mathbb{N}$ are the horizontal and vertical components of the pixel that represents the center of the camera aperture and (u_c, v_c) is known as the principle point. Finally, $r \triangleq \sqrt{\tilde{u}^2 + \tilde{v}^2}$ is the Euclidian distance in pixels between (u, v) and (u_c, v_c) where $\tilde{u} \triangleq u - u_c$ and $\tilde{v} \triangleq v - v_c$.

Before the distortion can be removed k_i, $i = 1, 2, 3$ and p_j, $j = 1, 2$ must be found. These coefficients are computed using the Camera Calibration Toolbox for Matlab ([59]) which uses the algorithms described in [60] to determine the distortion coefficients along with other camera specific parameters, such as the principle point. The toolbox uses a set of calibration images which are a series of pictures of a standard checkerboard training

pattern that is placed around the field of view of the camera. After the calibration images have been generated, they are loaded by the toolbox and the user selects the four outer corners of the pattern. After these corners have been selected, the toolbox finds the pattern intersections, where four of the squares on the pattern meet, in each image. Using a camera model ([61]) along with the physical properties of the calibration pattern, square size and number of rows and columns, the toolbox performs a Maximum Likelihood estimation of the camera parameters that minimizes the reprojection error in each of the intersection locations. After the distortion coefficients have been found, the distortion effects are removed from an image acquired by the camera using the OpenCV ([62]) function `cv::remap()` which removes the distortion by remapping each pixel in the image using the camera model and camera parameters. Once each pixel has been remapped, the new image matches what would be expected if the camera performed according the pinhole camera model and this allows us to calculate distances according to (3.4).

3.5.2. Image Segmentation

The sensors described in ([4, 5]) can only measure the distance to an object at a single point directly in front of the sensor. Our design takes advantage of laser line generators that project horizontal lines across the entire camera frame. By using laser lines instead of a single point we are able to measure distances at multiple locations. The ability to measure distances at multiple locations improves the sensor's ability to aid in mapping by providing richer information about an unknown environment, such as the shape of objects located in the environment. To calculate the distance at multiple locations the image is broken down into small segments as seen in Figure 3.8.

A secondary benefit of segmenting the image is that the line extraction component of the algorithm can be run on smaller images as opposed to the complete image. This provides

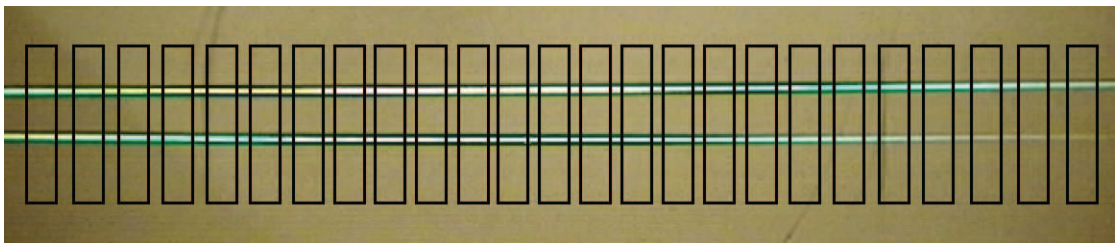


Figure 3.8: The segmentation process applied to a single video frame.

a performance benefit because processing times are shortened compared to the time that would be required to process the entire image.

3.5.3. Line Extraction

The line extraction component of the algorithm finds the location of the two laser lines in each image segment. By finding the vertical position of the two lines in each segment, the distance between the lines can be found which is the value that is needed to calculate the distance to the object in front of the camera. An overview of each of the steps used to extract the laser lines is shown in Figure 3.9.

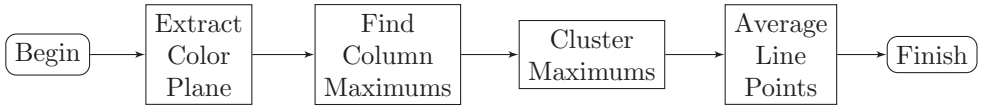


Figure 3.9: Overview of the Line Extraction Algorithm.

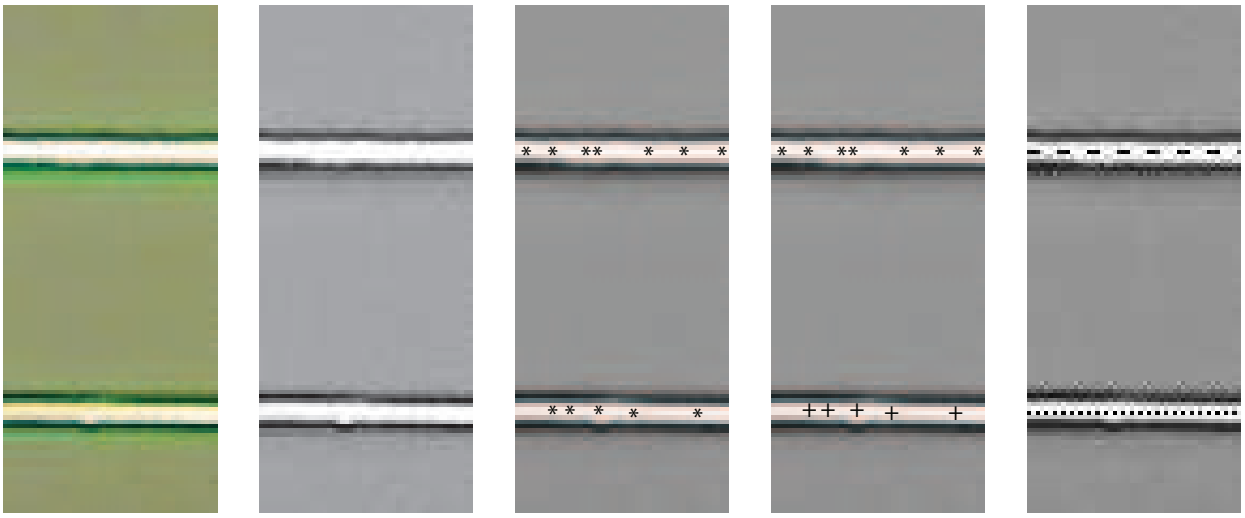
In the first step of the algorithm the green color is extracted from the image. The color plane extraction converts the image from color to black and white and the components of the original image that contained the largest amounts of green have the largest values in the extracted plane; these areas correspond to white in the black and white image. The extracted plane of Figure 3.10a can be seen in Figure 3.10b.

The laser lines run horizontal across the image so the pixels in each segment column with the largest values represent the points in that column with the largest amount of green and

we assume that they make up the laser line. To increase the speed at which the algorithm runs, not all columns in an image segment are examined, instead a subset of $m \in \mathbb{N}$ columns are processed. Each of the m columns are searched and the $n \in \mathbb{N}$ maxima are extracted. Each of the extracted maxima are compared to a threshold value to ensure that the value is above some minimum, this is to ensure that the selected points have a minimum amount of green in an attempt to ensure that the selected points are part of the laser line. A view of the extracted maxima for the sample image can be seen in Figure 3.10c where the maxima are marked with a "*". Once the maximum values for a column have been extracted the set of points are partitioned into two groups, one for each laser line. The partitioning is performed using K-Means Clustering ([63]). K-Means Clustering partitions our set of mn vertical location values, $\mathbf{v} \in \mathbb{N}^{mn}$, into 2 partitions ($\mathbf{p}_i$, $i = 1, 2$) by minimizing

$$J(\mathbf{v}) = \sum_{i=1}^{2} \sum_{\mathbf{v}(j) \in \mathbf{p}_i} \parallel \mathbf{v}(j) - c_i \parallel^2, \tag{3.7}$$

where $J : \mathbb{N}^{mn} \to \mathbb{R}$ is the objective function being minimized, $\mathbf{v}(j)$ is vertical position of the point being checked, and $c_i \in \mathbb{R}$ is the mean of the ith partition. The result of the partitioning is shown in Figure 3.10d where the points composing each of the two laser lines are marked with a "*" and a "+" respectively. Once the two sets of points making up each laser line have been found, the vertical position of each laser is determined by calculating the mean vertical position of each point set. The final location for each of the laser lines are displayed in Figure 3.10e with the dashed line representing one of the laser lines and the dotted line representing the other. Finally, the distance to the object is found using the vertical distance that separates each of the laser lines and (3.4).



(a) (b) (c) (d) (e)

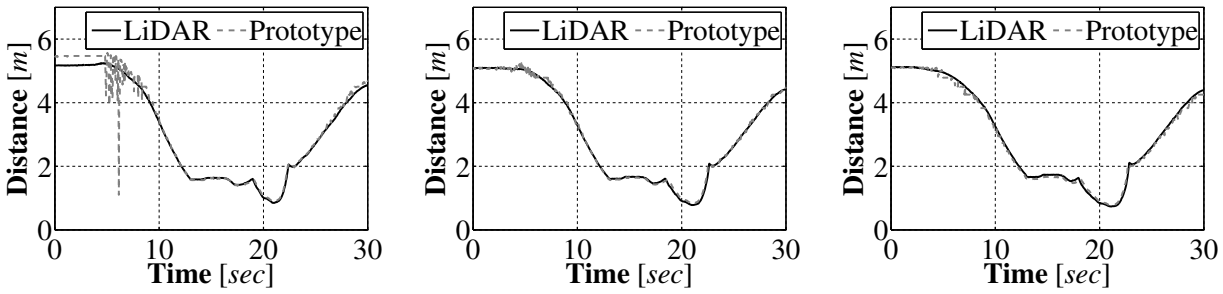
Figure 3.10: Each step of the laser line extraction algorithm.

3.6. Experimental Results

3.6.1. In Air Testing

To test the accuracy of our sensor, an environment was constructed in the laboratory using plywood. The distances found using the sensor prototype are compared to those measured with a Hokuyo UTM-30LX scanning laser range finder (LiDAR) which has an accuracy of $0.1-10$m ± 30mm, $10-30$m ± 50mm ([64]). The sensor prototype and LiDAR were attached to a wheeled cart and rolled through the test environment. The full image acquired by the camera was divided into 23 segments and a distance measurement was obtained for each segment. A comparison between the distances calculated using the sensor prototype and those measured with the LiDAR at three bearings can be seen in Figure 3.11.

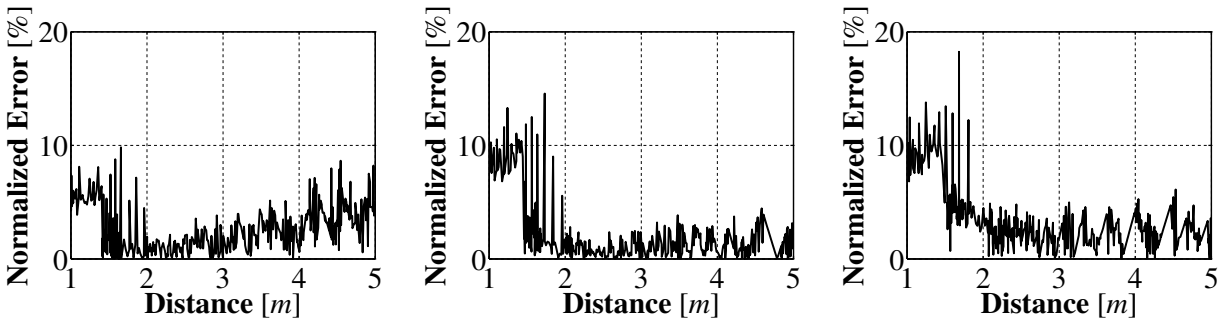
To better understand how well the sensor prototype measured the distance to objects, an error analysis was performed. The results of the error analysis for each of the bearings can be seen in Figure 3.12. From the analysis it can be seen that the measurement error as a percentage of the true distance, as measured by the LiDAR, is approximately 10% of the



(a) (b) (c)

Figure 3.11: Sensor prototype results vs. LiDAR over time at a relative bearing of −8° (a), 0° (b), and 8° (c).

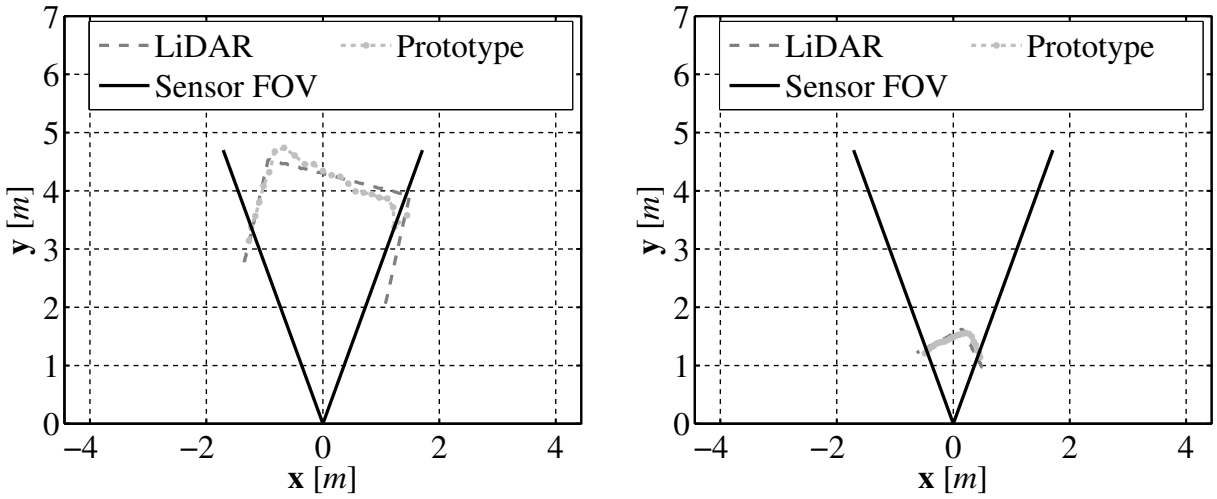
true distance. This result means that the closer the sensor is to an object the smaller the absolute error between our measured distance and the true distance.



(a) (b) (c)

Figure 3.12: Sensor error as a function of distance at a relative bearing of −8° (a), 0° (b), and 8° (c).

To show how shape information about an object can be found using our design, a set of frames are shown in Figure 3.13 that illustrate the advantage of using laser lines as opposed to a single laser point. When obstacles that are not flat and perpendicular to the camera's viewing axis are encountered, more information about the object is obtained with our sensor as opposed to the single distance to the object directly in front of the camera.



(a) (b)

Figure 3.13: Comparison of a corner as viewed by the prototype sensor and LiDAR from $4.5m$ (a) and $1.5m$ (b).

3.6.2. Underwater Testing

After the sensor was tested in the lab, a waterproof enclosure was constructed to allow for underwater testing. An outdoor, unfiltered test tank was used to test the sensor's ability to measure distances underwater. The sensor was held underwater facing one end of the test tank at 0.6m, 2.15m, and 4m. The results of the underwater test are shown in Figure 3.14 where the sensor is located at the origin pointed in the direction of the positive horizontal axis. Before testing the sensor, the camera calibration routine was run for the camera in the underwater environment to determine the distortion coefficients in water. The underwater testing results show that the sensor is capable of measuring the distance to an underwater object, with an estimated relative error close to the previously calculated 10%. From the results it can be seen that the sensor is able to determine the shape of objects in the underwater environment. In fact, the measurement at 4m captured both of the sides and the end of the test tank and the corners of the tank are easily seen. One drawback of our design is that since our sensor is based on the ability of the camera to see an object, more specifically

laser lines projected onto an object, any water conditions that negatively affect visibility will also negatively affect the sensor's performance.

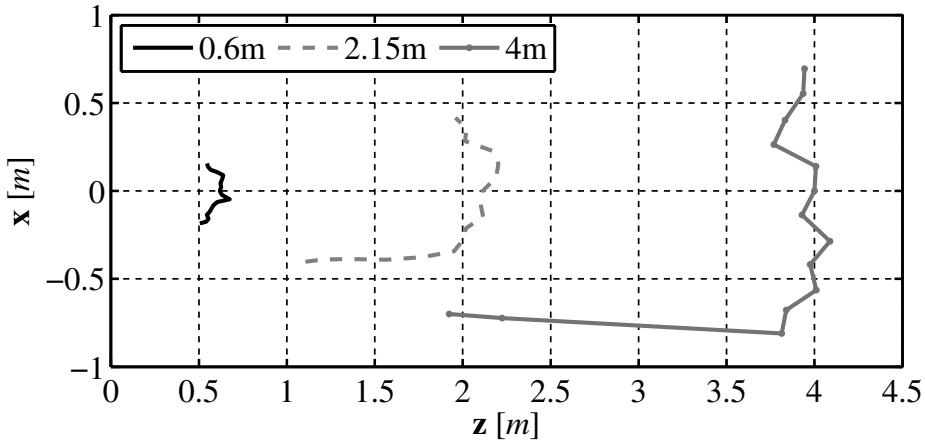


Figure 3.14: Experimental results when measuring distances underwater.

3.7. Error Analysis

By examining (3.4) with the assumption that the laser line generators can be mounted parallel to each other, the primary source of error in the distance measurement comes from the calculation of the distance that separates the laser lines in the camera image. To see how this error affects the distance measurement an error analysis was performed. By differentiating (3.4) the distance error is found to be

$$\delta z = -\tilde{y}\left(\frac{f}{\tilde{v}^2}\right)\delta\tilde{v}, \tag{3.8}$$

where $\delta z \in \mathbb{R}$ represents the distance error corresponding to laser line separation error $\delta\tilde{v} \in \mathbb{N}$. Equation (3.8) can be rewritten as

$$|\delta z| = \frac{f\tilde{y}}{\tilde{v}^2}|\delta\tilde{v}| \text{ or } |\delta z| = \frac{z^2}{f\tilde{y}}|\delta y|, \tag{3.9}$$

which shows that the absolute value of the measurement error grows quadratically with respect to the distance from the target object; this means that as the sensor moves further

away from an object the affect of laser line misidentifications becomes greater. Finally, using (3.4) we can rewrite (3.9) as

$$\frac{|\delta z|}{z} = \frac{|\delta \tilde{v}|}{\tilde{v}}, \tag{3.10}$$

which gives us the relationship seen in Figure 3.12 where the error as a percentage of *true* distance stays constant at approximately 10% of the distance.

3.8. Conclusions

In this chapter a low cost distance measuring sensor was designed and tested in both the air, where the accuracy of the sensor could be easily tested, and underwater. The low cost and ability to use waterproof equipment (lasers and cameras) make the proposed sensor useful for underwater robotics applications where a given level of visibility can be assured. The resulting sensor provides distance data with an error of 10% of the distance to the object which is accurate enough to be used on underwater vehicles moving at slow speeds. The sensor prototype also provides shape information about objects that can be useful for mapping and localization as well as simple obstacle avoidance. In the next chapter the implementation of a vision based odometry sensor is presented to complete the inexpensive vision based sensor package that is required for localization and mapping tasks of an Autonomous Underwater Vehicle (AUV).

Chapter 4

Visual Odometry with Downward Facing Camera

4.1. Introduction

For a robotic vehicle to work autonomously in unknown environments it must be able to locate objects of interest in the surrounding environment, however it is also import that the vehicle knows where it is located within the environment. There are a several families of sensors that robotic vehicles use to accomplish this task. Vehicles working outdoors with an unobstructed view of the sky often make use of Global Positioning Satellites (GPS) sensors. GPS sensors receive messages from all of the satellites that they observe in the sky, with each message containing the location of the satellite and time that the message was generated. Using these messages the sensor calculates the distance from the sensor to each satellite, and triangulates the location of the sensor on the earth. Sensors working on the same principle have been developed for use in indoor environments. For example, the Cricket Indoor Localization ([65]) provides accurate positioning information, with a position error as low as 1cm, by calculating the distance to beacons, indoor *satellites*, by calculating the time difference between a radio frequency (FR) and ultrasound signal sent from the same location. Another example of an indoor *GPS* is the Hagisonic Stargazer ([66]) which provides

localization information by triangulating itself using reflective infrared stickers placed on the ceiling which are observed by an upward facing camera with infrared light source, with a resulting localization error as low as 2cm. On wheeled robotic vehicles, encoders placed on the axels are a common sensor used for providing localization information. There are several types of encoders readily available, magnetic and optical being common, however the functionality is the same. Encoders measure the rotation of a shaft; by counting the number of axel rotations along with knowing the diameter of the vehicles wheels the distance that a wheeled vehicle has traveled can be estimated as well as the heading. Underwater vehicles typically use a Doppler Velocity Log (DVL) sensor to obtain information similar to that provided by encoders on wheeled ground vehicles ([67]). A DVL operates by facing downwards and bouncing an acoustic signal off of the operating environment floor, using the time that it takes the signal to return to the sensor, the speed of vehicle is determined.

Each of these sensor families suffer from issues that prohibit their use on our inexpensive underwater vehicle. GPS does not work underwater because the sensor is unable to receive the required messages from the GPS satellites, while a custom system based on the same idea similar to the indoor"GPS"systems discussed above could be developed, this would require modifications to the underwater environment which is undesirable. If the vehicle was driven by wheels, as many underwater crawlers are, then wheel axel encoders could be used to provide wheel rotation information. However, this information would be prone to large errors in an underwater environment where slippage in multiple directions is prevalent. DVL usage is promising for small vehicles, a new family of sensors such as Teledyne RD Instruments' Explorer ([68]) have been developed for small underwater vehicles. However, these sensors are expensive and designed to operate at a minimum distance of 0.5m off of the floor surface which makes their use impractical on a vehicle in constant contact with the bottom surface of the environment under inspection.

To overcome these issues, a downward facing camera to provide visual odometry (VO) information is considered. The use of downward facing cameras are quite common in many robotics applications due to their low cost and ease of use. In [15] an optical flow system is developed for a vehicle designed for planetary rover missions ([69, 70]). The design makes use of four optical mouse sensors, small low resolution cameras with computing chip, each with different focal length lenses to calculate two dimensional velocities in the vehicle's body frame. In [13] two separate cameras, one downward facing and one forward facing, are used to determine the optical flow information of a skid steered ground vehicle using the Lucas Kanade optical flow algorithm ([19]). A frequency domain image registration technique is developed in [71] for providing translation and rotation between subsequent image frames captured by a downward-facing camera. With background information on visual odometry provided an overview of the developed algorithm is provided next.

4.2. Visual Odometry Algorithm

The visual odometry algorithm that we developed is based on [14] which estimates vehicle translations using a downward facing camera. An overview of the complete algorithm is shown in Figure 4.1.

Before the translations of our UUV can be calculated, the image captured by the downward facing camera must be preprocessed. The original image captured at time step k, $\mathbf{I}_{k,o} \in \mathbb{R}^{w \times h}$ (Figure 4.2a) where $w \in \mathbb{N}$ is the width and $h \in \mathbb{N}$ is the height of the image, is converted from the full color space to greyscale, $\mathbf{I}_{k,o} \rightarrow \mathbf{I}_{k,bw} \in \mathbb{R}^{w \times h}$ (Figure 4.2b). This conversion is required for the remaining steps of the algorithm to work properly. A filter is then applied to $\mathbf{I}_{k,bw}$ which serves two purposes: i) through experimentation it was found that filtering the image made the system more robust to inconsistent lighting conditions and ii) the filter is required because the floor of the environment in which our vehicle is intended

to operate does not have a significant number of visually identifiable features when viewed without the filter. In order for the template matching approach to perform correctly, the image captured by the downward facing camera must have unique features that can be tracked. In some environments where our vehicle will operate, the floor has an almost uniform color while possessing an unique texture, similar to a poured concrete surface. This texture can be used to provide the unique features that are required and the filtering step makes the texture more apparent, as seen in Figure 4.2c.

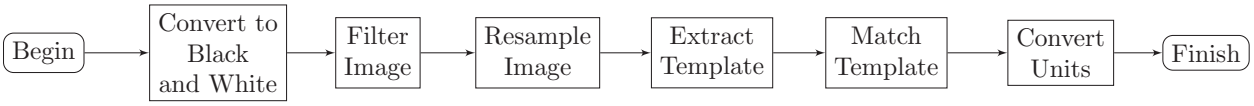
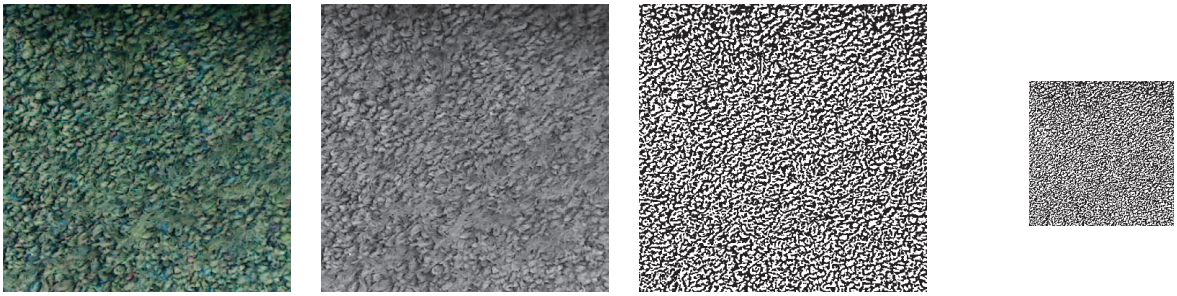


Figure 4.1: Overview of visual odometry algorithm.



(a) (b) (c) (d)

Figure 4.2: Each preprocessing step in the visual odometry algorithm.

The filter we apply is a Laplacian ([72]) and is defined as

$$\mathbf{L}(\mathbf{I}) \triangleq \frac{\partial^2 \mathbf{I}}{\partial u^2} + \frac{\partial^2 \mathbf{I}}{\partial v^2}, \tag{4.1}$$

where $\mathbf{I} \in \mathbb{R}^{w \times h}$ is a matrix of intensity values that make up the image and $\mathbf{L}(\mathbf{I}) \in \mathbb{R}^{w \times h}$ is the Laplacian of $\mathbf{I}$. Directly computing (4.1) can be computationally intensive so we approximate the Laplacian by convolving $\mathbf{I}_{k,bw}$ with a filter kernel ([72]). The filter kernel selected is a 7×7 Laplacian approximation kernel (Table 4.1).

Table 4.1: 7×7 Laplacian Approximation Kernel

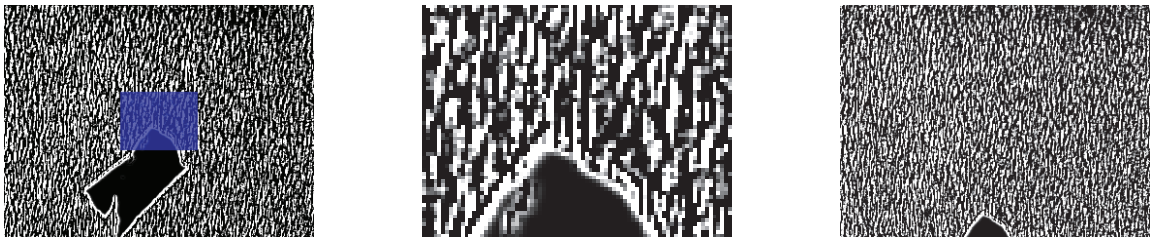
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	48	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

To decrease the feature tracking execution time, in the final preprocessing step, before the template matching is performed, we reduce the resolution of $\mathbf{I}_{k,bw}$ through resampling. In the resampling process, $\mathbf{I}_{k,bw} \rightarrow \mathbf{I}_{k,r} \in \mathbb{R}^{w_r \times h_r}$ where $w_r < w$ is the width of the resampled image and $h_r < h$ is the height. By reducing the resolution of the image we reduce the amount of data that must be processed during the tracking process, thus reducing the execution time. The result of resampling can be seen in Figure 4.2d.

Once the image has been preprocessed, the UUV translations can be calculated. The process that we use is referred to as template matching and involves finding the location of one image in another image. The first step of the template matching algorithm involves extracting a template image, $\mathbf{T}_k \in \mathbb{R}^{w_t \times h_t}$ where $w_t \in \mathbb{N}$ and $h_t \in \mathbb{N}$ are the width and height of the template, from $\mathbf{I}_{k-1,r}$. Our UUV moves at slow speeds so $\mathbf{T}_k$ is extracted from the center of $\mathbf{I}_{k-1,r}$. If our vehicle was moving faster it could prove advantageous to extract $\mathbf{T}_k$ from an alternate location that would give the best possible chance of $\mathbf{T}_k$ being present in $\mathbf{I}_{k,r}$. An example of the extraction location can be seen in Figure 4.3a along with the extracted template Figure 4.3b.

Once $\mathbf{T}_k$ has been extracted, the next step is finding the location of $\mathbf{T}_k$ in $\mathbf{I}_{k,r}$. The template matching process is performed by cross correlating $\mathbf{T}_k$ with $\mathbf{I}_{k,r}$ which yields

$$\mathbf{C}_k = \frac{1}{w_t h_t} \sum_{u,v} \frac{\left(\mathbf{I}_{k,r}\left(u,v\right) - \bar{I}_{k,r}\right)\left(\mathbf{T}_k\left(u,v\right) - \bar{T}_k\right)}{\sigma_{\mathbf{I}}\sigma_{\mathbf{T}}}, \tag{4.2}$$



(a) (b) (c)

Figure 4.3: The location of the template image (a), the extracted template used for tracking purposes (b), and the image that was searched (c).

where $\mathbf{C}_k \in \mathbb{R}^{w \times h}$ is the cross correlation matrix, $w_t h_t \in \mathbb{N}$ is the total number of pixels in $\mathbf{T}_k$, $\sigma_{\mathbf{I}}, \sigma_{\mathbf{T}} \in \mathbb{R}$ are the standard deviations and $\bar{I}_{k,r}, \bar{T}_k \in \mathbb{R}$ are the mean of the pixels in $\mathbf{I}_{k,r}$ and $\mathbf{T}_k$ respectively. The maximum value of $\mathbf{C}_k$ (Figure 4.4) is located at $(u_m, v_m) \in \mathbb{N}^2$ and corresponds to the center of $\mathbf{T}_k$ in $\mathbf{I}_{k,r}$.

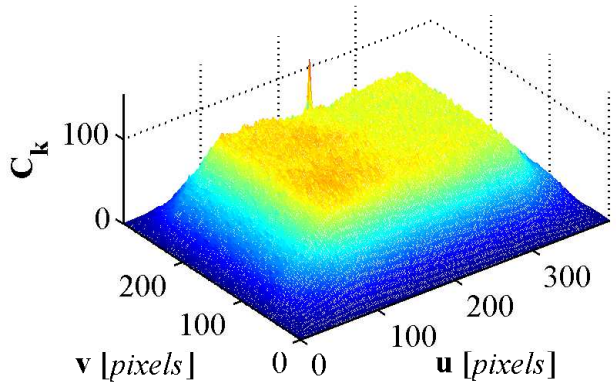


Figure 4.4: The cross correlation matrix $\mathbf{C}_k$ generated from the template matching step.

Once $\mathbf{T}_k$ is located in $\mathbf{I}_{k,r}$ the translations of the camera attached to the vehicle can be calculated using

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} u_m \\ v_m \end{bmatrix} - \begin{bmatrix} u_c \\ v_c \end{bmatrix}, \tag{4.3}$$

where $\delta u, \delta v \in \mathbb{N}$ are the translation of the camera, in pixels, in the horizontal and vertical direction and $(u_c, v_c) \in \mathbb{N}^2$ is the center of $\mathbf{I}_{k,r}$. These translations are not in physical units,

but rather in pixels, the camera's native units. To provide useful measurements for odometry purposes δu and δv are converted to physical units by

$$\begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = c \begin{bmatrix} \delta v \\ \delta u \end{bmatrix}, \tag{4.4}$$

where $c \in \mathbb{R}$ is a scaling factor that converts pixels to physical units and $\delta x, \delta y \in \mathbb{R}$ are the translations of the camera in the forward and lateral directions respectively.

Using δx and δy which are in the vehicle's body frame, the translations of the vehicle in a global frame of reference can be estimated. The translations in the global frame are found using an odometry model which assumes that the center of the vehicle is located at $(x_k, y_k) \in \mathbb{R}^2, k \geq 0$ and posses a global heading $\theta_k \in [-\pi, \pi), k \geq 0$. Using the vehicle translation in the body frame and a global heading provided by a compass, the translations in the global frame are given as

$$\begin{bmatrix} \Delta x_k \\ \Delta y_k \end{bmatrix} = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta y_k \end{bmatrix}, \tag{4.5}$$

where $\Delta x_k, \Delta y_k \in \mathbb{R}$ are the horizontal and vertical translation of the vehicle in the global frame. The global position of the vehicle is given by

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \end{bmatrix} + \begin{bmatrix} \Delta x_k \\ \Delta y_k \end{bmatrix}, \tag{4.6}$$

which when expanded using (4.5) yields the final odometry model for the system

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \end{bmatrix} + \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta y_k \end{bmatrix}. \tag{4.7}$$

4.3. Experimental Results

The visual odometry algorithm was tested to determine how well it performs. A downward facing camera was mounted to a small ground vehicle (Figure 4.6) for testing purposes

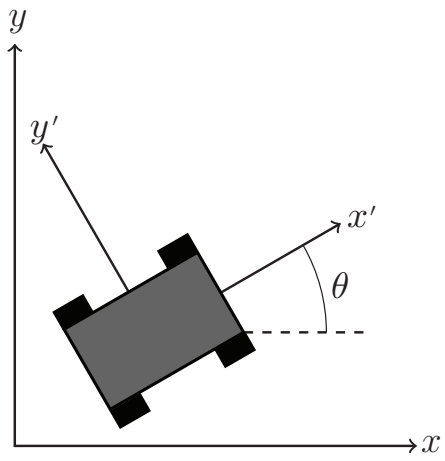


Figure 4.5: Overview of the visual odometry model.

along with a Hagisonic Stargazer indoor localization sensor ([66]) that was used to provide an experimental baseline. The vehicle was driven around a test environment logging position estimates provided by the visual odometry system as well as those provided by the Stargazer. The location estimates for an experimental run can be seen in Figure 4.7a and a plot of the corresponding error, using the Euclidean distance between the two estimates, is shown in Figure 4.7b. As seen in these results there is an error between the estimate produced by the visual odometry system and that provided by the Stargazer, which has a manufacturer reported error as low as 2cm. The error comes from the successive build up of small errors in the visual odometry measurements. Since each position estimate is based on the previous estimate, small errors at each time step build up over time, referred to as sensor *drift*. As discussed in the following chapter, this is not going to be a problem for the localization algorithm since the obstacle measurement provided by the laser range finder, will help to correct such a drift.

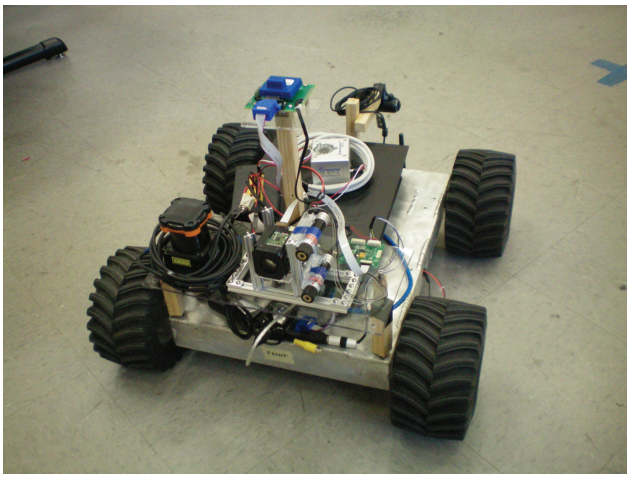
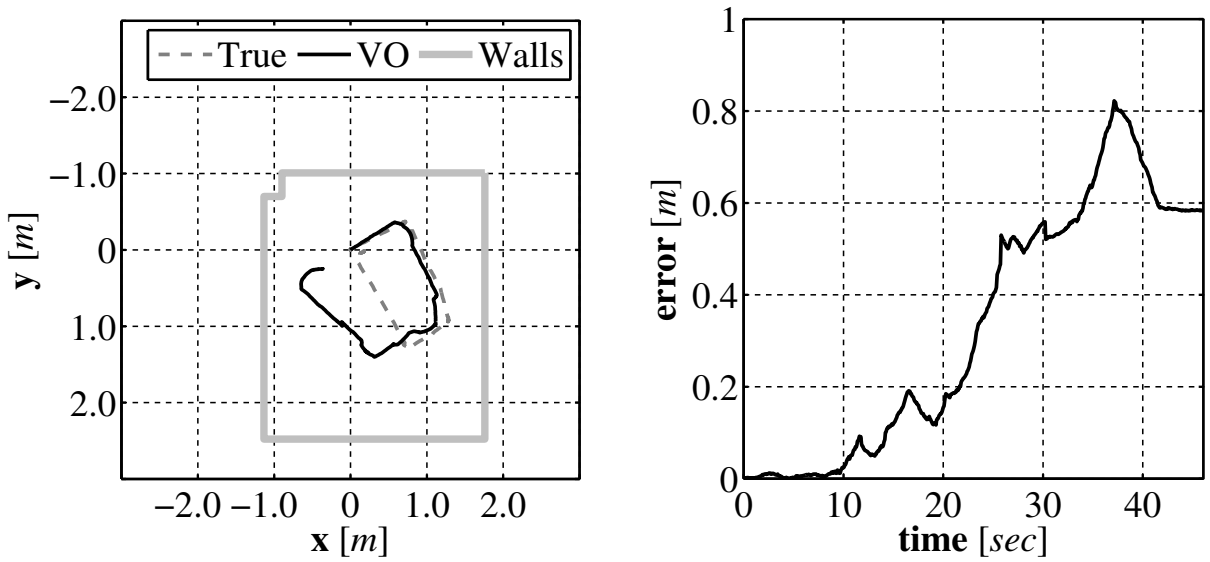


Figure 4.6: Test platform used for indoor tests.



(a) (b)

Figure 4.7: Visual odometry path estimate (a) and the error in the estimate as a function of time (b).

4.4. Conclusions

In this chapter a visual odometry algorithm was developed that makes use of a downward facing camera. The aim is to use the downward facing camera on a low cost UUV to provide localization information. In Section 4.2 the visual odometry algorithm was presented and experimental results using the system were presented in Section 4.3. With the visual odometry system and laser based rangefinder developed all of the sensors that are required to solve the SLAM solution have been developed or introduced. In the following chapter one of the *classic* solutions to the SLAM problem, the EKF SLAM algorithm, will be introduced and used to verify our assumption that the chosen suite of sensors provides information that is accurate enough to be used by our UUV to solve the SLAM problem.

Chapter 5

Sensor Validation with EKF SLAM

5.1. Introduction

In the previous two chapters a set of low cost sensors based on computer vision where developed for use on an UUV. Each sensor was independently tested to check its accuracy and performance. From the testing results it is believed that the sensors perform adequately for the task of localization and mapping. To verify this belief the Extended Kalman Filter (EKF) solution to the SLAM problem (EKF SLAM) was chosen as a verification algorithm to test the sensor package. The origins of the EKF SLAM algorithm, along with the solution presented in this chapter, are based around several important publications that were some of the original formulations of the SLAM problem, particularly [21] and [22]. The EKF SLAM algorithm, as the name implies, makes use the EKF ([73]) to solve the SLAM problem. The EKF is a generalization of the standard Kalman Filter ([74]) which is an implementation of the recursive Bayesian estimation solution to the SLAM problem introduced in Section 2.2.3. The EKF solution allows for nonlinearities to exist in the state transition and measurement models of the system and accounts for these nonlinearities by linearizing the models using a first order Taylor Series expansion.

This chapter is organized as follows. Section 5.2 introduces he SLAM problem that is

solved using EKF SLAM. Section 5.3 provides the details of the EKF SLAM algorithm. In Section 5.4 the feature extraction and data association procedure that are key to the EKF SLAM algorithm are discussed. Experimental results using the EKF SLAM algorithm are presented in Section 5.5 and concluding remarks are provided in Section 5.6.

5.2. The SLAM Problem

In our application, the SLAM problem can be thought of as providing an UUV with the ability to determine, when placed in an unknown environment, where in that environment it is located while building a map of the environment. The *online* SLAM problem, which EKF SLAM solves, attempts to estimate the pose our UUV and the map of the environment at the current time step k. In a probabilistic sense the online SLAM problem is attempting to find

$$p\left(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{5.1}$$

where the $\boldsymbol{\xi}_k = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix} \in \mathbb{R}^2 \times [-\pi, \pi)$ is the instantaneous pose of the vehicle, $\mathbf{M}$ is the map of the environment, $\mathbf{u}_{1:k}$ is the full series of controls performed by the vehicle, and $\mathbf{z}_{1:k}$ is the full series of observations collected. The SLAM problem is an example of a Markov chain and the standard solution to problems of this type is the recursive Bayesian estimator also referred to as the Bayes filter.

5.3. EKF SLAM Algorithm

The EKF is one of the earliest implementations of the Bayes filter. Before presenting the solution to the SLAM problem, a method of storing the map, $\mathbf{M}$ in (5.1), must be selected. In many instances it is easiest to think of the map as a set of discrete points in an environment. Each of these discrete locations in the environment are known as *landmarks* and are areas of

the environment that represent *features* that can be extracted from raw sensor data. Some common types of features that are used by robotic vehicles are walls, corners, or columns for vehicles operating in indoor environments. Using this approach the map is defined as

$$\mathbf{M} = \left\{ \begin{array}{cccc} \mathbf{m}_1, & \mathbf{m}_2, & \ldots, & \mathbf{m}_n \end{array} \right\}, \tag{5.2}$$

where $\mathbf{m}_i = (x_i, y_i) \in \mathbb{R}^2$ are the two dimensional Cartesian coordinates of the ith landmark in $\mathbf{M}$ and $i = 1, \ldots, n$ where $n \in \mathbb{N}$ is the total number of landmarks in $\mathbf{M}$. The family of SLAM solutions that represent the world using this type of map are known as feature based SLAM solutions. When landmarks are used to make up the map, an important component of SLAM is the ability to determine which landmark an observed feature represents. The feature to landmark relationship is represented using a set of correspondence values $\mathbf{c}_k \in \mathbb{N}^m$ where $m \in \mathbb{N}$ is the number of observations in $\mathbf{z}_k$. If the observation $\mathbf{z}_k(i)$ is generated because of the jth landmark then $\mathbf{c}_k(i) = j$.

The EKF SLAM algorithm estimates the pose of a vehicle and the map of the environment so the full state of the system being estimated is defined as

$$\mathbf{x}_k \triangleq \left[\begin{array}{cccc} \boldsymbol{\xi}_k & \mathbf{m}_1 & \mathbf{m}_2 & \cdots \quad \mathbf{m}_m \end{array} \right]^{\mathrm{T}}. \tag{5.3}$$

We assume that the full system behaves as

$$\mathbf{x}_k = \mathbf{g}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \boldsymbol{\epsilon}_k, \tag{5.4}$$

$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \boldsymbol{\delta}_k, \tag{5.5}$$

where $\mathbf{x}_k \in \mathbb{R}^{3+2n}$ is the state of the system, $\mathbf{z}_k \in \mathbb{R}^m$ is the current set of observations, and $\mathbf{u}_k \in \mathbb{R}^c$ is the current control input. The function $\mathbf{g} : \mathbb{R}^{3+2n} \times \mathbb{R}^c \to \mathbb{R}^n$ is the nonlinear state transition model that defines how the system evolves between time steps based on $\mathbf{x}_{k-1}$ and $\mathbf{u}_k$. The function $\mathbf{h} : \mathbb{R}^{3+2n} \to \mathbb{R}^m$ is the nonlinear measurement model and it describes

how $\mathbf{z}_k$ is related to $\mathbf{x}_k$. The variables $\boldsymbol{\epsilon}_k$ and $\boldsymbol{\delta}_k$ are additive zero mean Gaussian noise with covariances of $\mathbf{R}_k$ and $\mathbf{Q}_k$ respectively

$$\boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \mathbf{R}_k) \text{ and } \boldsymbol{\delta}_k \sim \mathcal{N}(0, \mathbf{Q}_k). \tag{5.6}$$

Using these assumptions, the EKF SLAM algorithm estimates the online SLAM posterior (5.1) as a Gaussian distribution

$$p(\boldsymbol{\xi}_k, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}) = p(\mathbf{x}_k \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}) = \mathcal{N}(\hat{\mathbf{x}}_k, \boldsymbol{\Sigma}_k), \tag{5.7}$$

where $\hat{\mathbf{x}}_k \in \mathbb{R}^{3+2n}$ is the mean vector of the estimate and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{(3+2n)\times(3+2n)}$ is the covariance matrix that describes the uncertainty in the estimate. The full algorithm that solves the online SLAM problem using the EKF is shown in Algorithm 5.1.

5.3.1. Prediction

The first step of the EKF SLAM algorithm is referred to as the prediction stage and is based on the state transition model (5.4) of the system, also referred to as the motion model, which describes how the full SLAM state evolves between time steps. The prediction step uses $\mathbf{g}(\cdot)$ to incorporate $\mathbf{u}_k$ into the estimate. A predicted mean vector is generated according to

$$\bar{\hat{\mathbf{x}}}_k = \mathbf{g}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k), \tag{5.8}$$

where $\bar{\hat{\mathbf{x}}}_k \in \mathbb{R}^{3+2n}$ is the predicted value of the mean vector. The state transition model $\mathbf{g}(\cdot)$ updates the vehicle pose using the motion model of the vehicle and $\mathbf{u}_k$. We assume that the environment in which the vehicle operates is static so $\mathbf{g}(\cdot)$ predicts the landmark locations using their estimated location at the previous time step $k-1$.

The use of noisy control inputs causes uncertainty to be added to the estimate, this uncertainty is incorporated in the second phase of the prediction step. The covariance

Algorithm 5.1 Overview of the feature based EKF SLAM algorithm.

procedure EKFSLAM($\hat{\mathbf{x}}_{k-1}, \mathbf{\Sigma}_{k-1}, \mathbf{u}_k, \mathbf{z}_k$)
    Data:
        $\hat{\mathbf{x}}_{k-1}$ the mean vector from the previous time step
        $\mathbf{\Sigma}_{k-1}$ the covariance matrix from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of observations

    Result:
        $\hat{\mathbf{x}}_k$ the current mean vector
        $\mathbf{\Sigma}_k$ the current covariance matrix

    $\bar{\hat{\mathbf{x}}}_k \leftarrow \mathbf{g}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k\right)$
    $\bar{\mathbf{\Sigma}}_k \leftarrow \mathbf{G}_k \mathbf{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}} + \mathbf{R}_k$
    $\mathbf{c}_k \leftarrow$ DataAssociate($\bar{\hat{\mathbf{x}}}_k, \mathbf{z}_k$)
    for $i = 1, \ldots, m$ **do**
        if $\mathbf{c}_k(i) \neq -1$ **then**
            $\mathbf{K}_k \leftarrow \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k\right)^{-1}$
            $\hat{\mathbf{x}}_k \leftarrow \bar{\hat{\mathbf{x}}}_k + \mathbf{K}_k \left(\mathbf{z}_k(i) - \mathbf{h}\left(\bar{\hat{\mathbf{x}}}_k\right)\right)$
            $\mathbf{\Sigma}_k \leftarrow (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\mathbf{\Sigma}}_k$
            $\bar{\hat{\mathbf{x}}}_k \leftarrow \hat{\mathbf{x}}_k$
            $\bar{\mathbf{\Sigma}}_k \leftarrow \mathbf{\Sigma}_k$
        end if
    end for
    for $i = 1, \ldots, m$ **do**
        if $\mathbf{c}_k(i) = -1$ **then**
            $\hat{\mathbf{x}}_k^+ \leftarrow \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{f}\left(\hat{\mathbf{x}}_k, \mathbf{z}_k(i)\right) \end{bmatrix}$
            $\mathbf{\Sigma}_k^+ = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{A}_k^{\mathrm{T}} \\ \mathbf{A}_k & \mathbf{B}_k \end{bmatrix}$
        end if
    end for
end procedure

matrix prediction increase the uncertainty in the estimate according to

$$\bar{\mathbf{\Sigma}}_k = \mathbf{G}_k \mathbf{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}} + \mathbf{R}_k, \tag{5.9}$$

where $\bar{\mathbf{\Sigma}}_k \in \mathbb{R}^{(3+2n)\times(3+2n)}$ is the predicted covariance matrix, $\mathbf{R}_k \in \mathbb{R}^{(3+2n)\times(3+2n)}$ is the covariance matrix of the state transition model, and $\mathbf{G}_k \in \mathbb{R}^{(3+2n)\times(3+2n)}$ is the Jacobian of $\mathbf{g}(\cdot)$ with respect to the system state.

The first term of the covariance prediction, $\mathbf{G}_k \mathbf{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}}$, propagates the uncertainty of the estimate from $k-1$ to k. The second term, $\mathbf{R}_k$, incorporates the additional uncertainty caused by the noisy control input. The landmark predictions do not cause any additional uncertainty to be added the system, only the control inputs add uncertainty, so $\mathbf{R}_k$ can be defined, using the covariance matrix of the control inputs, as

$$\mathbf{R}_k \triangleq \mathbf{V}_k \mathbf{M}_k \mathbf{V}_k^{\mathrm{T}}, \tag{5.10}$$

where $\mathbf{M}_k \in \mathbb{R}^{c\times c}$ is the covariance matrix of $\mathbf{u}_k$ and $\mathbf{V}_k \in \mathbb{R}^{(3+2n)\times c}$ is the Jacobian of $\mathbf{g}(\cdot)$ with respect to the control input. The full details of the EKF SLAM prediction step four our UUV can be seen in Section A.3.

5.3.2. Correction

The second step of the EKF SLAM algorithm is referred to as the correction stage. The correction stage uses the set of feature observations $\mathbf{z}_k$ and (5.5) to adjust the mean vector of the estimate while reducing the uncertainty contained in the covariance matrix. The mean vector correction is performed according to

$$\hat{\mathbf{x}}_k = \bar{\hat{\mathbf{x}}}_k + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{h}\left(\bar{\hat{\mathbf{x}}}_k \right) \right), \tag{5.11}$$

where $\mathbf{K}_k \in \mathbb{R}^{(3+2n)\times 2m}$ is the Kalman gain matrix. The Kalman gain matrix is a weighting matrix that creates a best estimate by defining how important the observation is when it is

incorporated, based on the covariance values of the state transition model and measurement model. The Kalman gain matrix is defined as

$$\mathbf{K}_k = \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}, \tag{5.12}$$

where $\mathbf{H}_k \in \mathbb{R}^{2m \times (3+2n)}$ is the Jacobian of $\mathbf{h}(\cdot)$ with respect to the system state and $\mathbf{Q}_k \in \mathbb{R}^{2m \times 2m}$ is the covariance matrix of the measurement model. The corrected covariance matrix of the estimate is generated according to

$$\mathbf{\Sigma}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\mathbf{\Sigma}}_k, \tag{5.13}$$

where $\mathbf{I}$ is a $3 + 2n$ identity matrix. The full details of the correction step for our UUV can be seen in Section A.4.

5.3.3. Augmentation

An additional step present in the EKF SLAM algorithm that does not belong to the standard EKF involves the addition of new landmarks to the estimate. As a UUV moves through an unknown environment new landmarks are found as unexplored areas are visited. When these features are observed for the first time the state vector and covariance matrix must be augmented to include the new landmarks. The mean vector augmentation is given by

$$\hat{\mathbf{x}}_k^+ = \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{z}_k(i)) \end{bmatrix}, \tag{5.14}$$

where $\hat{\mathbf{x}}_k^+ \in \mathbb{R}^{5+2n}$ is the mean with the newly observed landmark added and $\mathbf{f} : \mathbb{R}^{3+2n} \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is the inverse measurement model that calculates the landmark location in the global frame based on $\hat{\mathbf{x}}_k$ and $\mathbf{z}_k(i)$.

The augmentation of the covariance matrix is more complicated as $\mathbf{\Sigma}_k$ contains the covariance matrices of the vehicle pose estimate and the landmark location estimates along

with the cross covariance terms that relates each element in $\hat{\mathbf{x}}_k$ to every other element. The covariance matrix augmentation is given by

$$\boldsymbol{\Sigma}_k^+ = \begin{bmatrix} \boldsymbol{\Sigma}_k & \mathbf{A}_k^{\mathrm{T}} \\ \mathbf{A}_k & \mathbf{B}_k \end{bmatrix}, \tag{5.15}$$

where $\boldsymbol{\Sigma}_k^+ \in \mathbb{R}^{(5+2n)\times(5+2n)}$ is the covariance matrix following the augmentation. The matrix $\mathbf{A}_k \in \mathbb{R}^{2\times(3+2n)}$ is defined as

$$\mathbf{A}_k \triangleq \mathbf{F}_{k,x}\boldsymbol{\Sigma}_k, \tag{5.16}$$

where $\mathbf{F}_{k,x} \in \mathbb{R}^{2\times 3+2m}$ is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the system state and it propagates the uncertainty in the estimate before augmentation into the new feature cross covariance terms. The matrix $\mathbf{B}_k$ is defined as

$$\mathbf{B}_k \triangleq \mathbf{F}_{k,x}\boldsymbol{\Sigma}_k\mathbf{F}_{k,x}^{\mathrm{T}} + \mathbf{F}_{k,z}\mathbf{Q}_k\mathbf{F}_{k,z}^{\mathrm{T}}, \tag{5.17}$$

where $\mathbf{F}_{k,z} \in \mathbb{R}^{2\times 2}$ is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the current observation. The matrix $\mathbf{B}_k$ takes the current uncertainty and adds the uncertainty caused by the observation sensors to generate the full uncertainty in the location estimate for the new feature. The details of the augmentation step are given in Section A.5 for our system.

5.4. Feature Extraction and Data Association

EKF SLAM builds a map of the world using a set of landmarks. In order to use these landmarks, a key aspect in implementing the algorithm is developing a method of extracting features that correspond to the landmarks from raw sensor data. Based on the sensor being used, almost anything in the environment can be used for landmarks. In our testing environment corners are used because the environment is a simple rectangular room so identifying corners is simple. A feature extraction algorithm was written to extract corners from the range and bearing measurements returned by our laser based range finder developed in Chapter 3. The algorithm is a modified version of the Random Sample Consensus

(RANSAC) algorithm ([75]) for line identification. In the standard RANSAC line identification algorithm, a random set of points are selected then a best fit line is generated through those points. The full set of points are compared to that line and if a minimum number of points lie on the line then that line assumed to be a true line in the sensor data. Our sensor has very few points and can be quite noisy at larger distance so randomly selecting points to create a line led to a large number of invalid walls being identified. In our algorithm a single point in the sensor data is selected at random and all points that fall within a certain distance of that point are used to generate the potential line.

Once all lines in the laser range finder data are found using the RANSAC algorithm, each line is compared to every other and if two lines have an angle between them that is larger than a minimum corner angle the intersection of those two lines is identified as a corner. An example of the corner identification is show in Figure 5.1.

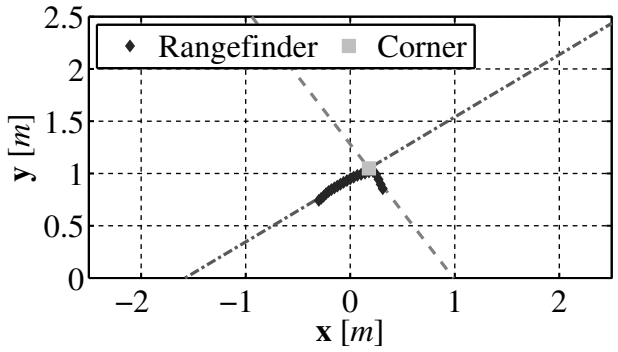


Figure 5.1: Corner identification using the modified RANSAC algorithm.

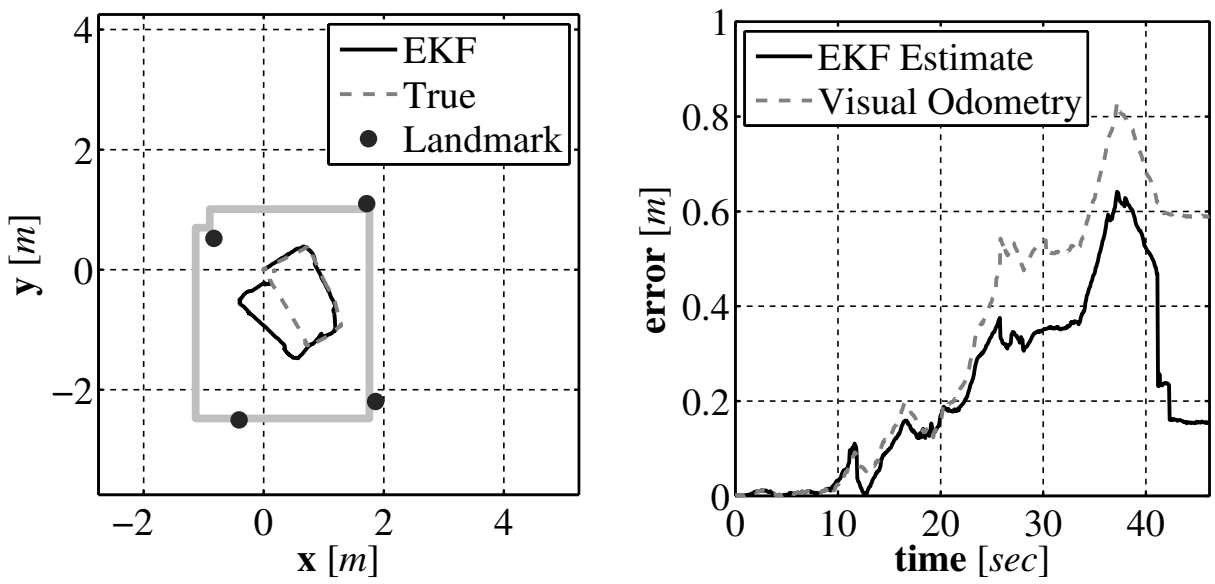
The second key component in the implementation of EKF SLAM is data association, previously discussed in Section 5.3.2. Data association involves finding which landmark in $\hat{\mathbf{x}}_k$ corresponds to each observed feature in $\mathbf{z}_k$. If a given observed feature in $\mathbf{z}_k$ corresponds to a landmark in $\hat{\mathbf{x}}_k$ then the estimate is corrected using the observed feature as seen in Section 5.3.2. If the observed feature does not correspond to any landmark then the newly observed feature is used to augment $\hat{\mathbf{x}}_k$ as described in Section 5.3.3. In our test environment

the minimum distance between corners was quite large, $>$ 1m, so a simple search algorithm was developed to perform the data association. At time step k, a global location is generated for each of the observed features, $\mathbf{z}_k(i), i = 1, 2, \ldots, m$, using $\bar{\hat{\mathbf{x}}}_k$ and $\mathbf{h}(\cdot)$. If the location of $\mathbf{z}_k(i)$ is within some maximum distance of the jth landmark in $\bar{\hat{\mathbf{x}}}_k$ then $\mathbf{c}_k(i) = j$. If no corresponding landmark is found for the ith observed feature then $\mathbf{c}_k(i) = -1$.

5.5. Experimental Results

In order to examine how well the selected sensor suite can perform mapping and localization, each of the proposed sensors (laser based rangefinder, downward facing camera, and compass) was attached to the test platform described in Section 4.3. In order to provide a location baseline for comparison purposes, a Stargazer indoor localization sensor was also attached to the test platform. The platform was driven remotely around an indoor test environment that measured 3m $\times$3.5m while executing EKF SLAM. The final vehicle path and map are shown in Figure 5.2a and the position error during the run is shown in Figure 5.2b. The position error is calculated using the euclidian distance between the estimate and baseline. From the results it can be seen that the error in the estimate produced by EKF SLAM never exceeds 0.7m. To illustrate how the estimate and the uncertainties change during the run a sequence of images are shown in Figure 5.3 that display the estimate and uncertainties.

It can be seen in the sequence that initially the uncertainty in the position estimate, represented by a 2σ covariance ellipse, is very small and the uncertainty in the first landmark estimate, which is initially observed before moving, is also small. As the vehicle moves through the environment the uncertainty in the position estimate grows due to the noisy control signals. Due to the growing position uncertainty, the uncertainty in newly observed features also increases. The SLAM algorithm limits the growth of the uncertainty while the



(a) (b)

Figure 5.2: The EKF SLAM produced path and map estimate (a) and the error in the position estimate over time (b).

vehicle is moving, however the true benefit of the EKF SLAM algorithm can be seen at the end of the run when the robot returns near the starting position and observes the landmark it observed from the starting location. The robot had a good idea about the location of that feature so EKF SLAM uses this observation to correct all of the estimates that are maintained by the estimate. This correction, referred to as loop closure in the literature ([53]), not only updates the mean of the estimate for each component it also reduces the uncertainty of each estimate. To show how the uncertainty in the estimate changes over time Figure 5.4 displays the error in the position estimate of the UMV in the horizontal and vertical direction along with the 2σ boundary that indicates the level of uncertainty in the estimate. In a similar fashion Figure 5.5 shows the error in the horizontal position of the landmark estimates maintained by the map along with the 2σ boundary and Figure 5.6 shows the error in the vertical direction.

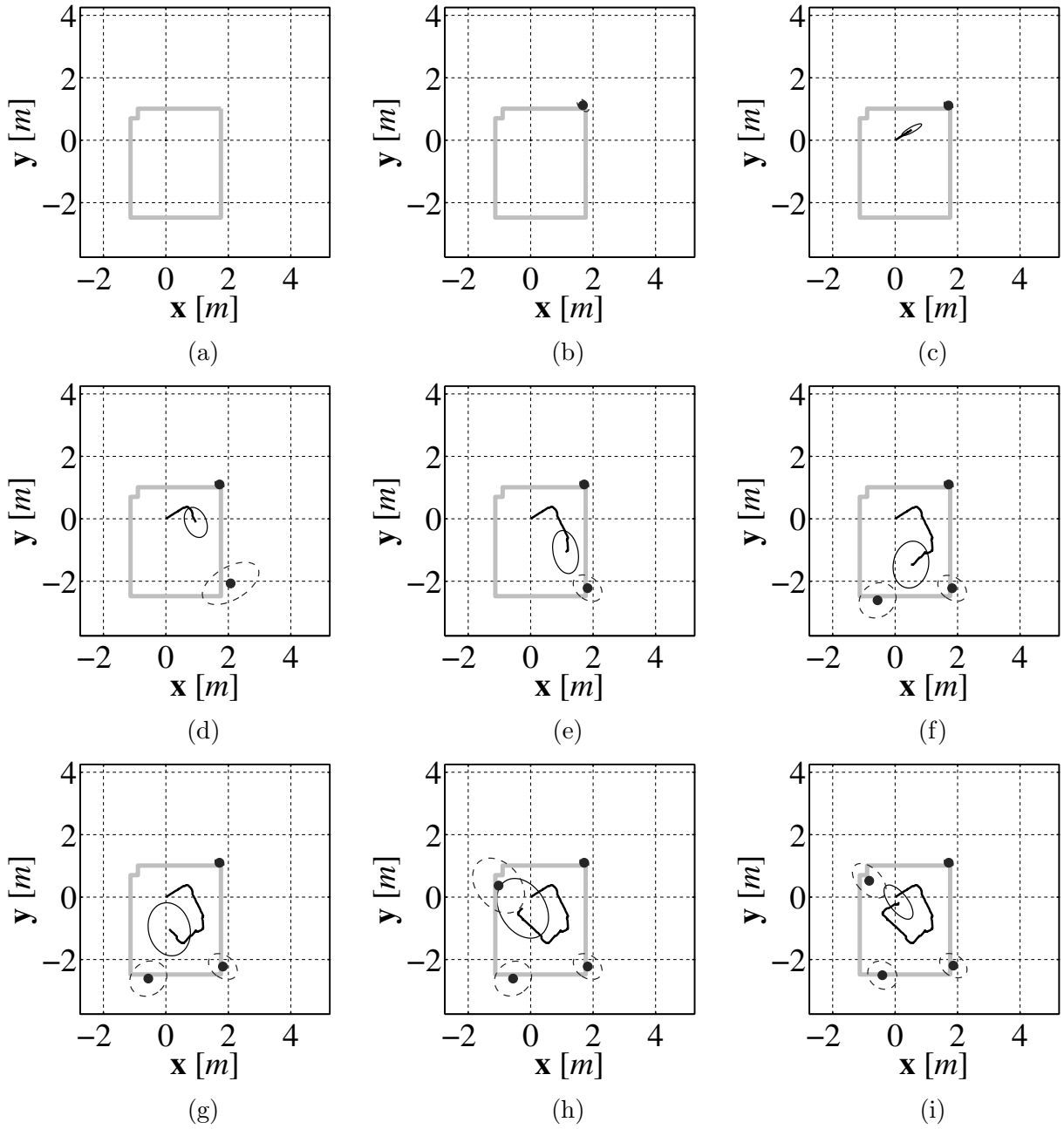


Figure 5.3: An evenly spaced sequence of images through out the entire localization and mapping process.

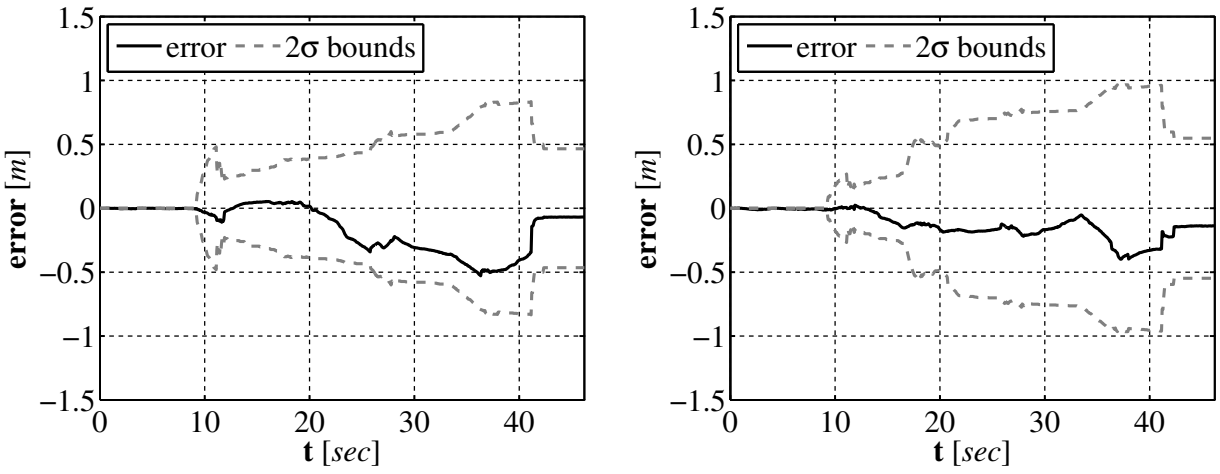


Figure 5.4: The error in the EKF SLAM position estimate in the horizontal (a) and vertical direction (b) over time along with the 2σ boundary.

5.6. Conclusions

In this chapter a classic localization and mapping algorithm, EKF SLAM, was used to validate a set of inexpensive sensors designed to provide localization and mapping functionality to a low-cost UUV. From the previously discussed results it can be seen that the sensor suite provides sufficient information for localization and mapping tasks. However, the EKF SLAM algorithm relies on a significant assumption about the noise characteristics of the sensors. The EKF SLAM approach assumes that the additive noise to the state transition model and measurement model is distributed according to a Gaussian distribution. While the noise characteristics of most sensors is not Gaussian the assumption has been shown to be appropriate for many sensors, include our inexpensive sensor suite. However, in an attempt to obtain results using more accurate sensor models a second family of SLAM algorithms exist that relax the Gaussian noise assumption. This family of algorithms we be investigated using our sensor suite in the following chapter.

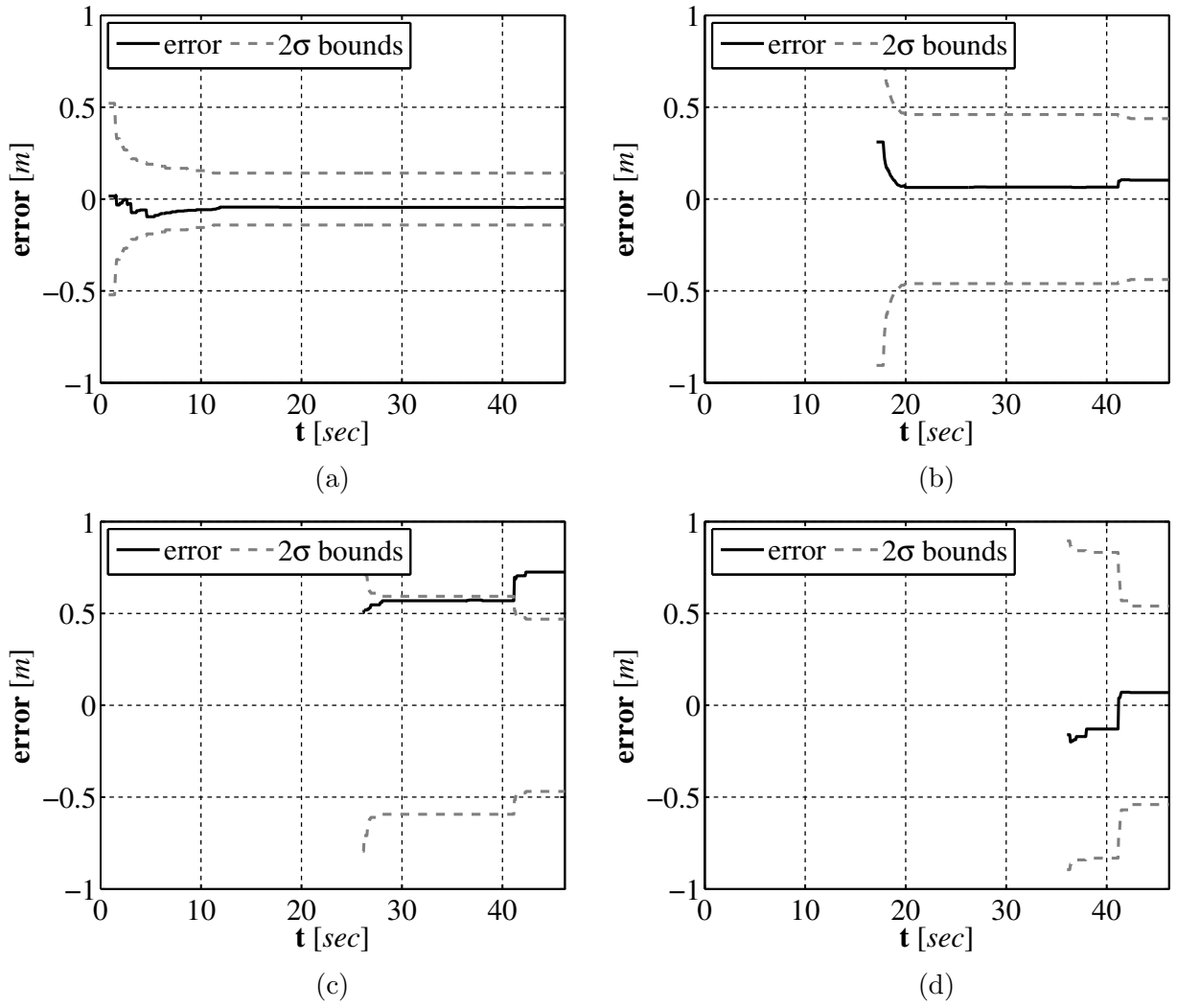


Figure 5.5: The error in the EKF SLAM horizontal landmark position estimate over time along with the 2σ boundary.

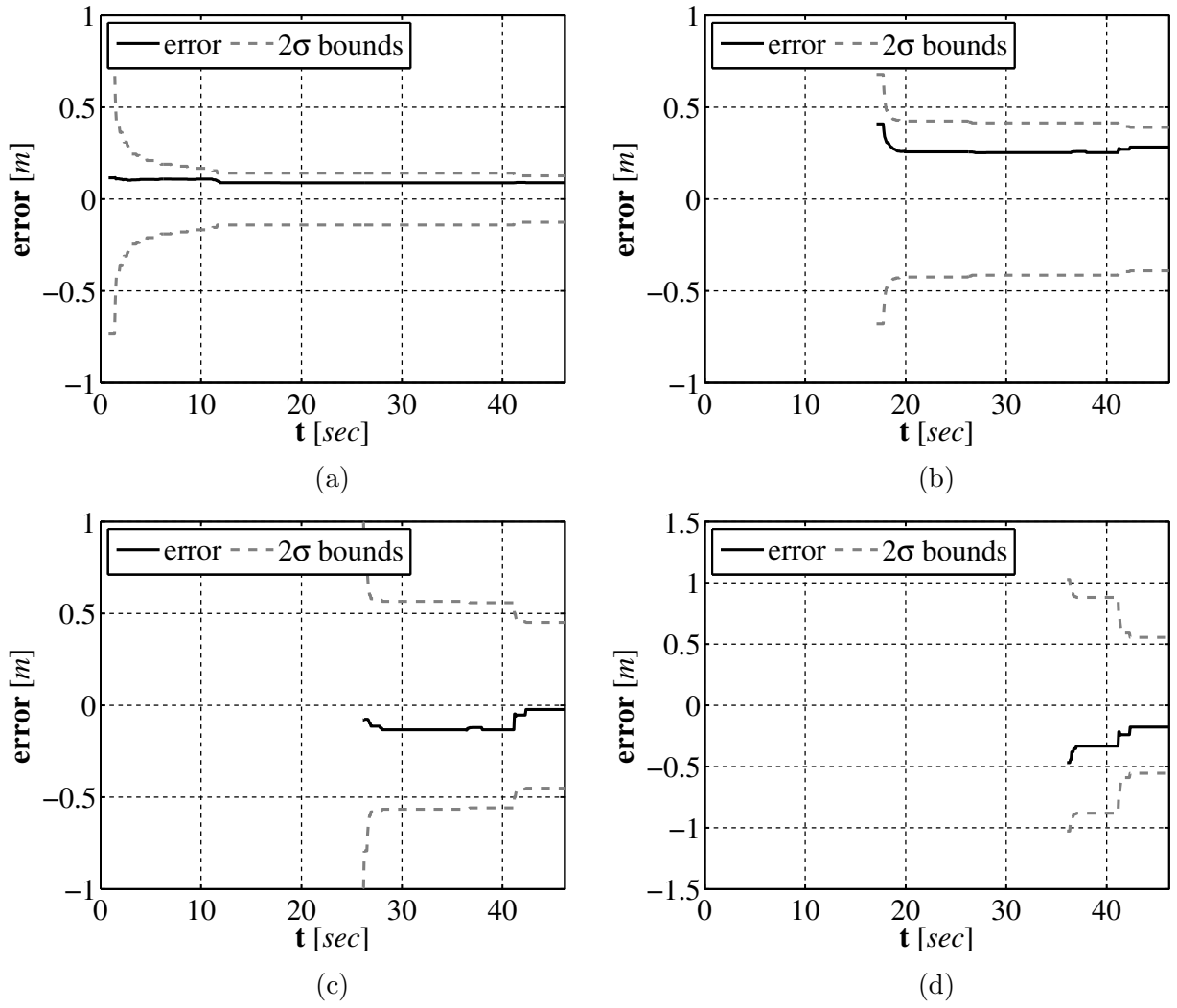


Figure 5.6: The error in the EKF SLAM vertical landmark position estimate over time along with the 2σ boundary.

Chapter 6

Sensor Validation with FastSLAM

6.1. Introduction

Following the EKF SLAM validation of the vision based sensors (Chapter 5) a second validation method was developed for examining the ability of the sensor suite to provided sufficient data for localization and mapping. Unlike the family of solutions, which the EKF SLAM solution is a member, that model the state estimate as a Gaussian distribution a second family of solutions exist that do not make this assumption. This alternate family of solutions provide the advantage of removing the requirement for additive Gaussian noises to the prediction and measurement models of the system which is advantageous as the noise present in most real world sensors is not Gaussian. By allowing the use of more realistic probabilistic sensor models these non-gaussian solutions have the potential of providing more accurate estimates of the SLAM posterior. The solution that was selected to provide a second set of experimental results for validation purposes is referred to as FastSLAM 1.0 ([33]), however for the remainder of this dissertation we will just refer to it as FastSLAM.

The remainder of this chapter is organized as follows. The FastSLAM algorithm is presented in Section 6.2. The method in which new landmarks are added to the map estimate is discussed in Section 6.3. The method of extracting features from the sensor data and

determining which landmarks they correspond to is presented in Section 6.4. Experimental results using our proposed sensor suite along with the FastSLAM algorithm are presented in Section 6.5 and concluding remarks are given in Section 6.6.

6.2. FastSLAM Algorithm

As opposed to the EKF solution to the SLAM problem that estimates the distribution that represents the instantaneous pose of a UMV and the map, the FastSLAM solution estimates the distribution that represents the full trajectory of the UMV and the map, this distribution is given by

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right). \tag{6.1}$$

By estimating the distribution that represents the full trajectory of the UMV, (6.1) can be factored into a pair of terms that are easier to estimate. Using the property of conditional independence and making the correspondence value explicit, as with the EKF SLAM approach, the factorization of (6.1) is given as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right) p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right). \tag{6.2}$$

Similarly to the EKF SLAM algorithm, the FastSLAM algorithm uses a feature based map. This map representation and the above decomposition allow us to factor (6.1) as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right) \prod_{i=1}^{n} p\left(\mathbf{m}_i \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right), \tag{6.3}$$

where the full map posterior is replaced by the product of landmark posteriors. The FastSLAM algorithm estimates the joint posterior, (6.3), using a Rao-Blackwellized particle filter which is an example of a sampling importance resampling (SIR) ([76]) particle filter and it is this type of filter that forms the basis of the FastSLAM algorithm. In the FastSLAM algorithm the distribution that represents the trajectory of the UUV is estimated using a

particle filter where each particle maintains its own copy of the map. The map maintained by the FastSLAM algorithm is composed of a set of n landmarks where the distribution corresponding to the estimate of each landmark is assumed to be a Gaussian distribution and estimated using an EKF. As a result each particle maintains a UMV pose estimate and n EKFs which represent the distribution corresponding to the set of landmark estimates with a mean vector and covariance matrix. The pth particle in the particle set $\mathcal{X}_k$ is defined as

$$\mathcal{X}_k^{[p]} \triangleq \left[\begin{array}{cccccc} \boldsymbol{\xi}_k^{[p]} & \hat{\mathbf{m}}_{k,1}^{[p]} & \boldsymbol{\Sigma}_{k,1}^{[p]} & \cdots & \hat{\mathbf{m}}_{k,n}^{[p]} & \boldsymbol{\Sigma}_{k,n}^{[p]} \end{array} \right], \tag{6.4}$$

where $\hat{\mathbf{m}}_{k,j}^{[p]} \in \mathbb{R}^2$, $j = 1, \ldots, n$ and $\boldsymbol{\Sigma}_{k,j}^{[p]} \in \mathbb{R}^{2\times 2}$, $j = 1, \ldots, n$ are the mean and covariance of the jth landmark estimate. For p particle and n landmarks the FastSLAM algorithm maintains pn EKFs each used to estimate a single landmark location. The FastSLAM algorithm is performed in a four-step procedure and an overview of the algorithm can be seen in Algorithm 6.1 and Algorithm 6.2.

6.2.1. Pose Sampling

The first step of the FastSLAM algorithm is referred to as pose sampling. In this step a set of potential particles, $\overline{\mathcal{X}}_k$, is generated from the set of particles, $\mathcal{X}_{k-1}$, that resulted from the previous time step. The set of potential particles is generated by sampling a new pose for each particle in $\mathcal{X}_{k-1}$ using the probabilistic motion model of the vehicle

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right). \tag{6.5}$$

Unlike EKF SLAM where the uncertainty introduced into the estimate is maintained by the covariance matrix, FastSLAM introduces uncertainty through the sampling process. The probabilistic motion model introduces uncertainty by adding noise to the control inputs that is distributed according to the probabilistic model of the control noise. In our implementation the control noise is assumed to be zero mean Gaussian with a covariance matrix $\mathbf{M}_k$; this

Algorithm 6.1 Overview of the feature based FastSLAM algorithm.

procedure FastSLAMOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
  Data:
    $\mathcal{X}_{k-1}$ the particles from the previous time step
    $\mathbf{u}_k$ the current control input
    $\mathbf{z}_k$ the current set of observations

  Result:
    $\mathcal{X}_k$ the new set of particles generated at the current time step

  $\overline{\mathcal{X}}_k \leftarrow \{\}$
  for $p \leftarrow 1, P$ **do**
    $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
    $\mathbf{c}_k^{[p]} \leftarrow \text{DataAssociate}(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k)$
    for $i = 1, \ldots, n$ **do**
      if i is not in $\mathbf{c}_k^{[p]}$ **then**
        $\hat{\mathbf{m}}_{k,i}^{[p]} \leftarrow \hat{\mathbf{m}}_{k-1,i}^{[p]}$
        $\boldsymbol{\Sigma}_{k,i}^{[p]} \leftarrow \boldsymbol{\Sigma}_{k-1,i}^{[p]}$
      end if
    end for
    $w_k^{[p]} = 1$
    for $i = 1, \ldots, m$ **do**
      if $\mathbf{c}_k(i) \neq -1$ **then**
        $\overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \hat{\mathbf{m}}_{k-1,\mathbf{c}_k(i)}^{[p]}$
        $\overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \boldsymbol{\Sigma}_{k,\mathbf{c}_k(i)}^{[p]}$
        $\mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_{k,\mathbf{c}_k(i)}^{\mathrm{T}} \left(\mathbf{H}_{k,\mathbf{c}_k(i)} \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_{k,\mathbf{c}_k(i)}^{\mathrm{T}} + \mathbf{Q}_k\right)^{-1}$
        $\hat{\mathbf{m}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]} + \mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \left(\mathbf{z}_k(\mathbf{c}_k(i)) - \mathbf{h}\left(\overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]}\right)\right)$
        $\boldsymbol{\Sigma}_{k,\mathbf{c}_k(i)}^{[p]} = \left(\mathbf{I} - \mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_k\right) \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]}$
        $w_k^{[p]} = w_k^{[p]} \times \eta \frac{1}{\sqrt{\left|2\pi \mathbf{Q}_k^{[p]}\right|}} \exp\left(-\frac{1}{2} \frac{\left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)^{\mathrm{T}} \left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)}{\mathbf{Q}_k^{[p]}}\right)$
      end if
    end for

Algorithm 6.2 Overview of the feature based FastSLAM algorithm Part2.

```
        for i = 1, ..., m do
            if c_k(i) = −1 then
                m̂^[p]_{k,n+1} ← f(ξ^[p]_k, z_k(j))
                Σ^[p]_{k,n+1} ← F_{k,z} Q_k F^T_{k,z}
            end if
        end for
        X̄^[p]_k ← [ ξ^[p]_k   M^[p]_k   w^[p] ]
    end for
    X_k ← {}
    for p = 1, n do
        select j with probability proportional to w^[p]
        X^[p]_k ← X̄^[j]_k
    end for
end procedure
```

form of control noise is not a requirement as opposed to EKF SLAM. This is one of the advantages of FastSLAM, more realistic noise models can be used as opposed to EKF SLAM that requires the control noise to be Gaussian. The only requirement for FastSLAM is the distribution that represents the noise on the control input must be of a form that can be sampled from.

The set of potential particles generated during the pose sampling step, $\overline{\mathcal{X}}_k$, are distributed according to

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}, \mathbf{c}_{1:k-1}\right), \tag{6.6}$$

which is denoted the *proposal distribution*. However, the proposal distribution does not match the distribution of particles that we are attempting to estimate, the first half of (6.3), which is referred to as the *target distribution* and given as

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right). \tag{6.7}$$

It can be seen that the distribution that we sample new particle poses from does not include the current observations at time step k. The current observation is incorporated into the

estimate in the third step of the algorithm, the importance weight calculation after which the set of potential particles, distributed according to (6.6), are transformed to approximate (6.7) during the final step of the algorithm, the resampling stage. The specific details of the pose sampling procedure described in this section can be found in Section B.2.

6.2.2. Feature Location Estimation

In the second step of the FastSLAM algorithm the landmark estimates maintained by each particle are updated using the current set of observations at time step k. Each particle maintains a set of n EKFs representing the particle's estimate for each landmark. The correspondence for each feature observation is calculated as described in Section 5.4 using $\boldsymbol{\xi}_k^{[p]}$ and $\mathbf{z}_k$. If the jth landmark is not observed at time step k, then the particle's estimate of the feature location remains unchanged

$$\hat{\mathbf{m}}_{k,j}^{[p]} = \hat{\mathbf{m}}_{k-1,j}^{[p]} \quad \text{and} \quad \boldsymbol{\Sigma}_{k,j}^{[p]} = \boldsymbol{\Sigma}_{k-1,j}^{[p]}. \tag{6.8}$$

If the jth landmark is observed at time step k, then the landmark estimate is updated using the standard EKF update equations, more specifically the correction equations, that make use of $\mathbf{h}(\cdot)$, the measurement model of the system (5.5) from Chapter 5, and $\mathbf{Q}_k$, the covariance matrix of the measurement model. As with the EKF SLAM algorithm, we assume that the landmarks used to represent the environment are static so the predicted mean and covariance are

$$\overline{\hat{\mathbf{m}}}_{k,j}^{[p]} = \hat{\mathbf{m}}_{k-1,j}^{[p]} \quad \text{and} \quad \overline{\boldsymbol{\Sigma}}_{k,j}^{[p]} = \boldsymbol{\Sigma}_{k,j}^{[p]} \ . \tag{6.9}$$

where $\overline{\hat{\mathbf{m}}}_{k,j}^{[p]} \in \mathbb{R}^2$ is the predicted mean of the jth landmark estimate and $\overline{\boldsymbol{\Sigma}}_{k,j}^{[p]} \in \mathbb{R}^{2\times 2}$ is the predicted covariance matrix of the estimate. The Kalman gain matrix, $\mathbf{K}_{k,j}^{[p]} \in \mathbb{R}^{2\times 2}$, is generated according to

$$\mathbf{K}_{k,j}^{[p]} = \overline{\boldsymbol{\Sigma}}_{k,j}^{[p]} \mathbf{H}_{k,j}^{\mathrm{T}} \left(\mathbf{H}_{k,j} \overline{\boldsymbol{\Sigma}}_{k,j}^{[p]} \mathbf{H}_{k,j}^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}, \tag{6.10}$$

where $\mathbf{H}_k \in \mathbb{R}^{2\times 2}$ is the Jacobian of $\mathbf{h}(\cdot)$ with respect to the landmark position. The landmark estimate is corrected using

$$\hat{\mathbf{m}}_{k,j}^{[p]} = \overline{\hat{\mathbf{m}}}_{k,j}^{[p]} + \mathbf{K}_{k,j}^{[p]} \left(\mathbf{z}_k(i) - \mathbf{h}\left(\overline{\hat{\mathbf{m}}}_{k,j}^{[p]} \right) \right), \tag{6.11}$$

and the covariance matrix of the landmark estimate is corrected according to

$$\mathbf{\Sigma}_{k,j}^{[p]} = \left(\mathbf{I} - \mathbf{K}_{k,j}^{[p]} \mathbf{H}_k \right) \overline{\mathbf{\Sigma}}_{k,j}^{[p]}, \tag{6.12}$$

where $\mathbf{I}$ is a 2 dimensional identity matrix. The details of the landmark location estimate updates can be found in Section B.3.

6.2.3. Importance Weight Calculation

As discussed in Section 6.2.1, the set of temporary particles that is generated in the sampling process, $\overline{\mathcal{X}}_k$, are distributed according to (6.6) which only includes the control input at time step k. However, the true distribution that we are attempting to estimate, (6.7), makes use of the current control input along with the current observation and set of correspondences. To overcome the difference between the two distributions an importance weight for each particle is generated. The form of the importance weight comes from the fact that the Rao-Blackwellized particle filter is version of a SIR particle filter. From [77] when we are unable to directly sample from the distribution that we are attempting to estimate, by considering the following importance weight for each particle

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \tag{6.13}$$

and particles are drawn with replacement from $\overline{\mathcal{X}}_k$ and added to $\mathcal{X}_k$ with a probability proportional to $w_k^{[p]}$, then $\mathcal{X}_k$ will approximate the target distribution and the quality of the approximation will improve as the number of particles increases.

From [33] the importance weight for the ith particle in $\overline{\mathcal{X}}_k$ is the ratio of the target distribution and proposal distribution

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right)}{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}, \mathbf{c}_{1:k-1}\right)} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right). \tag{6.14}$$

In order to calculate (6.14), we take note that the measurement $\mathbf{z}_k$ is also dependent on the map $\mathbf{M}$. Using this information (6.14) is expanded as

$$w_k^{[p]} = \eta p\left(\mathbf{z}_k \;\middle|\; \mathbf{M}, \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right) p\left(\mathbf{M} \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right). \tag{6.15}$$

The map is composed of a set of landmarks so we integrate over all landmarks and (6.15) becomes

$$w_k^{[p]} = \eta \int p\left(\mathbf{z}_k \;\middle|\; \mathbf{m}_{\mathbf{c}_k}, \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right) p\left(\mathbf{m}_{\mathbf{c}_k} \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right) d\mathbf{m}_{\mathbf{c}_k}. \tag{6.16}$$

Using the fact that each landmark estimate is dependent on the vehicle trajectory, all feature observations, and all landmark correspondences, (6.16) is written as

$$w_k^{[p]} = \eta \int p\left(\mathbf{z}_k \;\middle|\; \mathbf{m}_{\mathbf{c}_k}, \boldsymbol{\xi}_k^{[p]}, \mathbf{c}_k\right) p\left(\mathbf{m}_{\mathbf{c}_k} \;\middle|\; \boldsymbol{\xi}_{1:k-1}^{[p]}, \mathbf{z}_{1:k-1}, \mathbf{c}_{1:k-1}\right) d\mathbf{m}_{\mathbf{c}_k}. \tag{6.17}$$

This can be calculated in closed form as a Gaussian according to

$$w_k^{[p]} = \eta \frac{1}{\sqrt{\left|2\pi \mathbf{Q}_k^{[p]}\right|}} \exp\left(-\frac{1}{2} \frac{\left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)^{\mathrm{T}} \left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)}{\mathbf{Q}_k^{[p]}}\right), \tag{6.18}$$

where

$$\mathbf{Q}_k^{[p]} = \mathbf{H}_k^{\mathrm{T}} \boldsymbol{\Sigma}_{k-1,j}^{[p]} \mathbf{H}_k + \mathbf{Q}_k, \tag{6.19}$$

and where $\boldsymbol{\Sigma}_{k-1,j}^{[p]}$ is the covariance of the landmark estimate from $k-1$, $\mathbf{H}_k$ is the Jacobian of $\mathbf{h}(\cdot)$, and $\mathbf{Q}_k$ is the covariance matrix of the observation. The details of the importance weight calculation can be found in Section B.4.

6.2.4. Resampling

The final step of FastSLAM is resampling during which p particles are drawn with replacement from $\overline{\mathcal{X}}_k$ with a probability proportional to $w^{[p]}$ and added to $\mathcal{X}_k$. This step converts $\overline{\mathcal{X}}_k$ which is distributed according to (6.6) to the final particle set $\mathcal{X}_k$ which is distributed according to (6.7). The specific details of the algorithm we use to perform the resampling can be seen in Section B.5.

6.3. New Feature Addition

Similarly to the EKF SLAM algorithm, when features are observed that do not correspond to already tracked landmarks, a new landmark must be added to the map. From [54], when a new feature is observed the mean of the new landmark's estimate is initialized according to

$$\hat{\mathbf{m}}_{k,n+1}^{[p]} = \mathbf{f}\left(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(j)\right), \tag{6.20}$$

where $\mathbf{f} : \mathbb{R}^3 \times \mathbb{R}^2 \to \mathbb{R}^2$ is the inverse of $\mathbf{h}(\cdot)$ and it generates a landmark location based on a particle pose and a measurement. The covariance is initialized according to

$$\boldsymbol{\Sigma}_{k,n+1}^{[p]} = \mathbf{F}_{k,z}\mathbf{Q}_k\mathbf{F}_{k,z}^{\mathrm{T}}, \tag{6.21}$$

where $\mathbf{F}_{k,z}$ is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the observation and $\mathbf{Q}_k$ is the covariance of the measurement noise. The details of the addition of new landmarks to the map in the FastSLAM algorithm can be seen in Section B.6.

6.4. Feature Extraction and Data Association

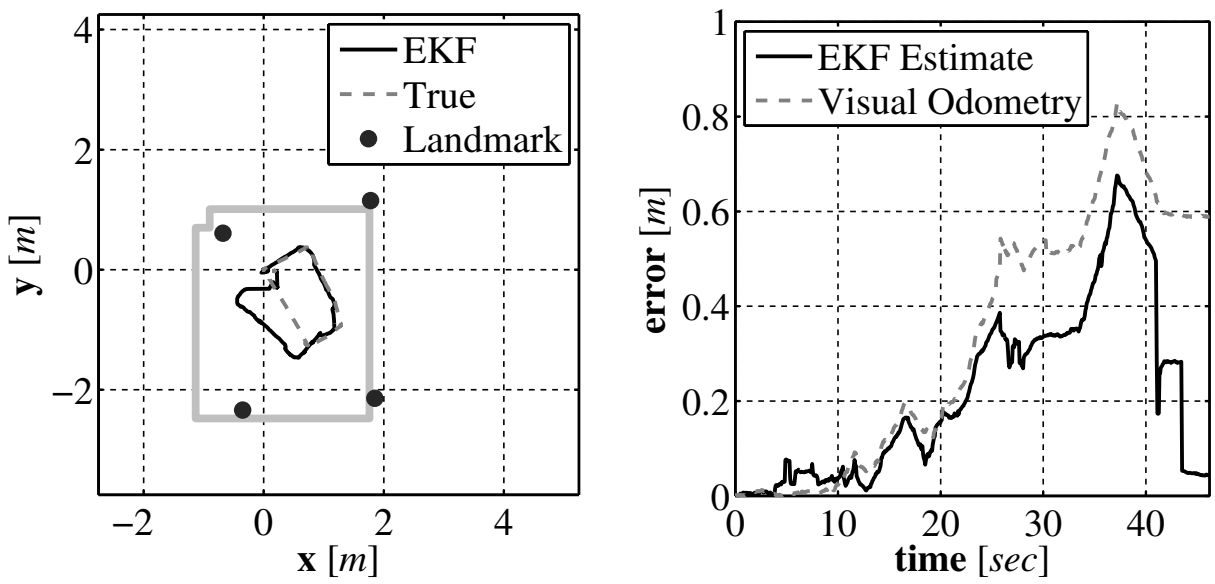
Just like the EKF SLAM algorithm, the FastSLAM algorithm uses a map composed of landmarks so a feature extraction and data association method must be selected. For the

following experimental results the same feature extraction and data association method used by the EKF SLAM algorithm and described in Section 5.4 was used.

6.5. Experimental Results

As with the EKF SLAM algorithm, validation testing is performed using the entire sensor suite. In particular, the laser rangefinder, downward facing camera, and compass were mounted to a test vehicle and driven around a test environment, the same vehicle and environment used in Section 4.3. The generated path estimate and map can be seen in Figure 6.1a compared to the path estimate provided by the Stargazer. The resulting path error as a function of time can be seen in Figure 6.1b. Since the vehicle posterior is represented by a set of particles, the mean of the particle set is used for display purposes and calculating the error. As seen in the error plot the position error never exceeds $0.7m$. To illustrate the FastSLAM process an equally spaced sequence of frames over the entire run are displayed in Figure 6.2.

In Figure 6.2 a 2σ covariance ellipse is shown along with the mean position estimate from the particle set along with the mean position estimate of each of the landmark locations. The performance of the FastSLAM algorithm is very close to that of the EKF SLAM approach seen in Section 5.5. The uncertainty is low for both the position and landmark estimates at the beginning of the run, described by the uncertainty ellipse that is generated using the covariance of the particle set. As the vehicle moves through the environment, the uncertainty grows until the vehicle returns near the starting location and re-observes the first landmark. At this time the particles that have positions nearest to the true position of the vehicle have very large importance weights so they are selected at higher probability than those farther away from the vehicles true position. This corrects the position estimate and brings the estimate much closer to the true position while also significantly reducing the spread of the



(a) (b)

Figure 6.1: The FastSLAM produced path and map estimate (a) and the error in the position estimate over time (b).

particle set. To illustrate the uncertainty in the UMV pose estimate Figure 6.3 displays the error in the position estimate the horizontal and vertical direction along with the 2σ boundary that indicates the level of uncertainty in the estimate and Figure 6.4 and Figure 6.5 displays the error in the horizontal and vertical direction of the landmark position estimates.

6.6. Conclusions

In this chapter a second feature based SLAM algorithm was used to validate the proposed sensor suite that has been designed for an inexpensive UUV. The algorithm has shown that the sensor suite performs adequately for the required mapping and localization process. Along, with validating the sensor suite the FastSLAM algorithm was used to address some of the issues of using the EKF SLAM algorithm that was previously described (Chapter 5). Primarily, the noise model of the state transition equation is relaxed from the Gaussian

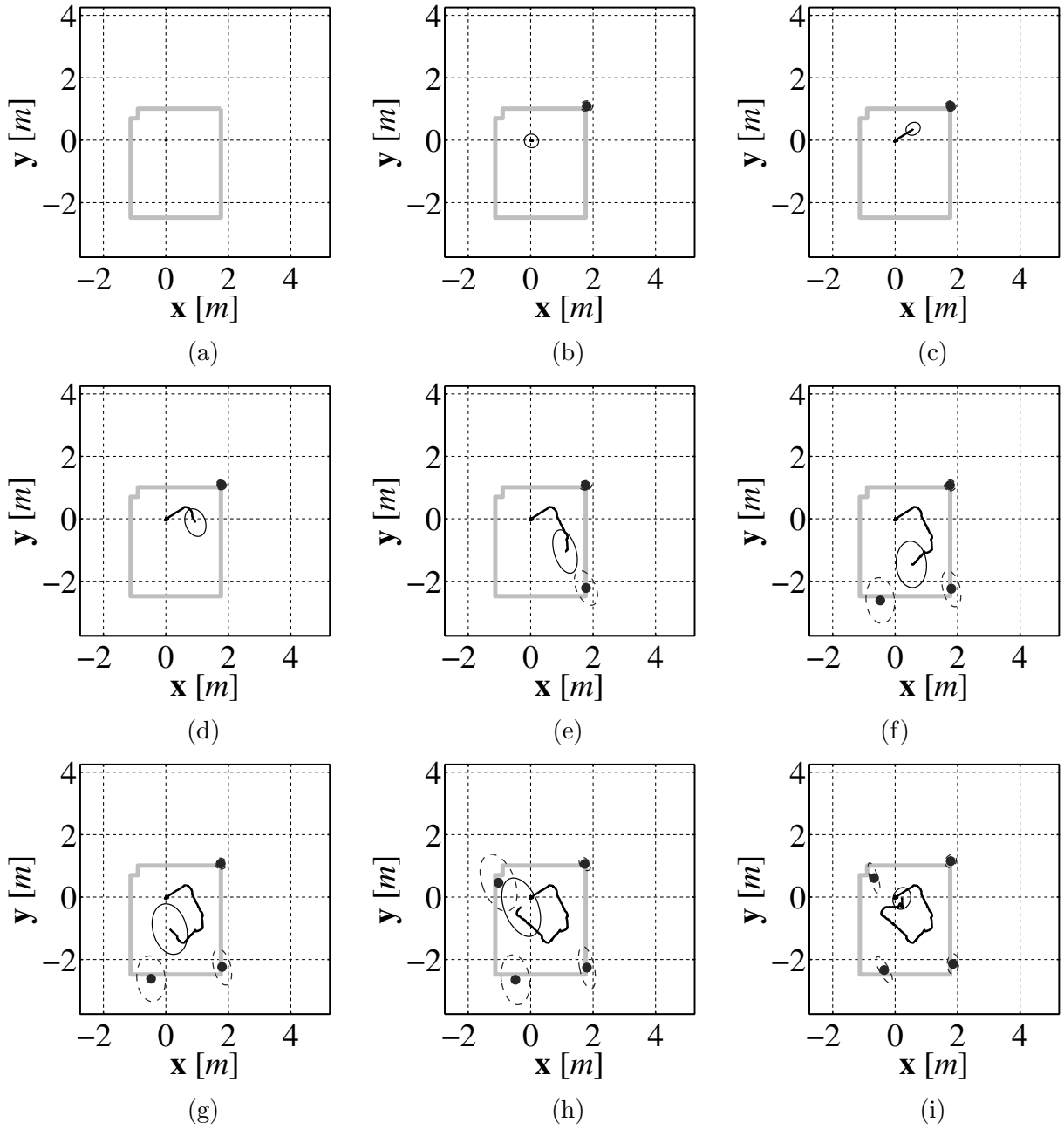
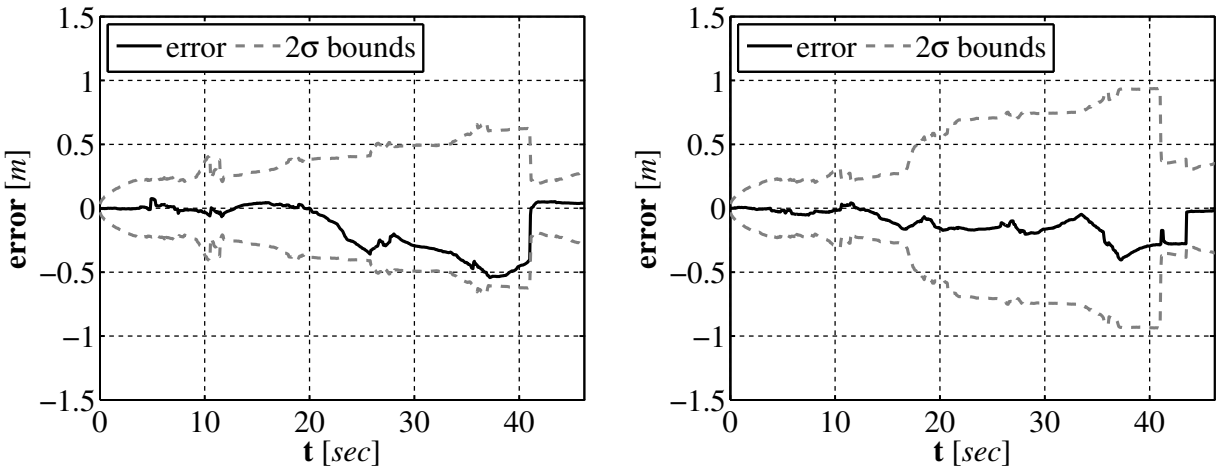


Figure 6.2: Evenly spaced sequence of path and map estimate produced by the FastSLAM algorithm.



(a) (b)

Figure 6.3: The error in the FastSLAM position estimate in the horizontal (a) and vertical direction (b) over time along with the 2σ boundary.

assumption made in the EKF SLAM algorithm to the only requirement being that the noise model is of a form that the FastSLAM algorithm can sample from it. However, the algorithm is still a feature based SLAM algorithm which places a requirement on the structure of the environment in which the vehicle operates. Also, each particle in the particle filter uses n EKFs to estimate the location of the n landmarks used to represent the environment. As a consequence the assumption still stands that the noise on the sensors used to locate features in the environment is Gaussian. In order to address these issues an extension of the FastSLAM algorithm that generates a map of the environment using occupancy grids is presented in the following chapter. This extension removes the restriction that the environment must be composed of landmarks and it removes the Gaussian noise assumption from the observation sensors.

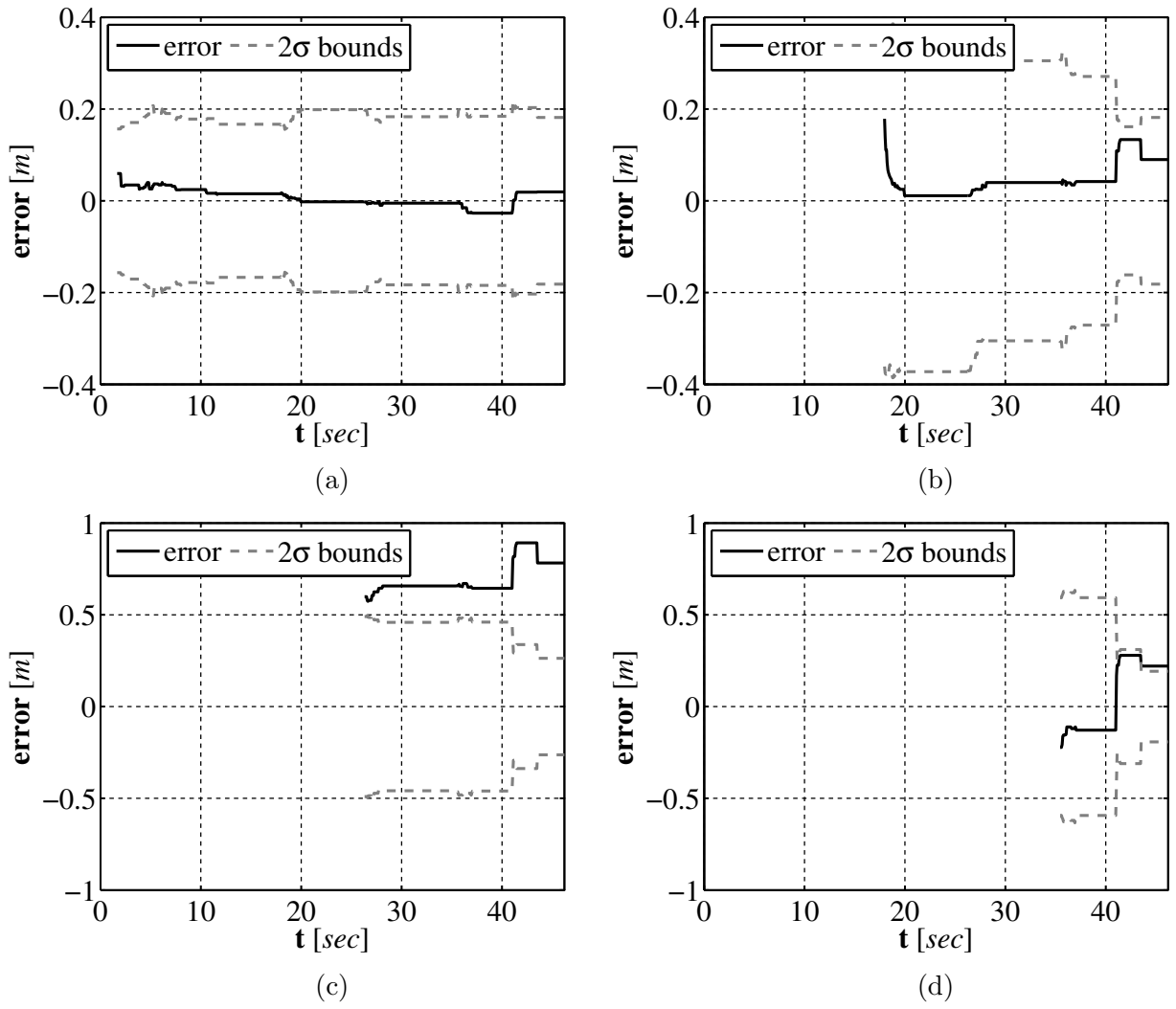


Figure 6.4: The error in the FastSLAM horizontal landmark position estimate over time along with the 2σ boundary.

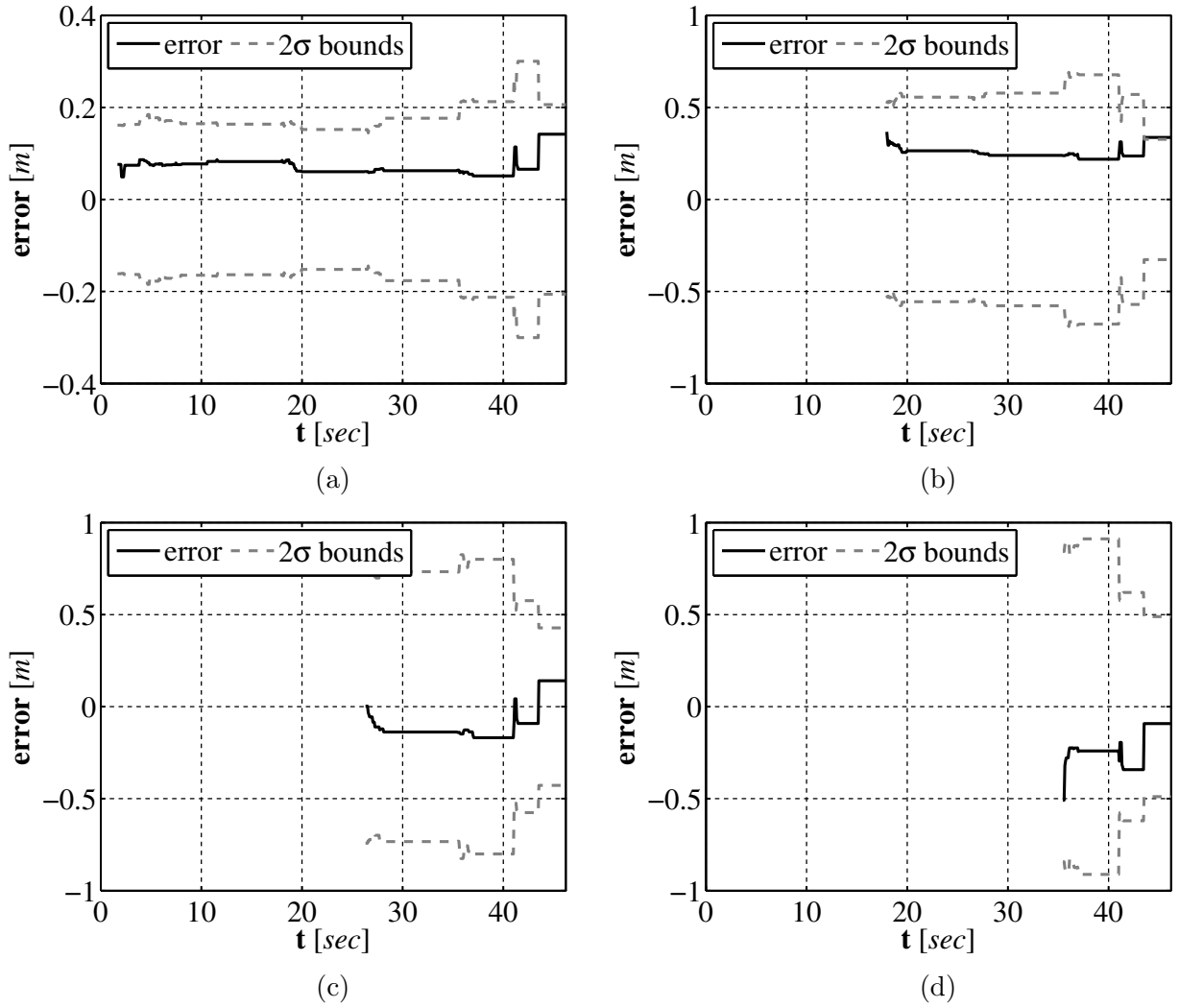


Figure 6.5: The error in the FastSLAM vertical landmark position estimate over time along with the 2σ boundary.

Chapter 7

FastSLAM With Occupancy Grids

7.1. Introduction

In the previous two chapters a pair of SLAM algorithms, EKF SLAM and FastSLAM, were used to validate our selected sensor suite. Each of these approaches represent the map of the environment in which a UMV operates using a set of discrete landmarks. This approach requires the environment to be structured in such a way that the sensors used by the UMV can identify these landmarks. Many environments in which UMVs operated are structured in a way that makes this approach difficult. Particularly in the underwater domain environments such as circular tanks, underwater cave systems, or submerged structures such as shipwrecks are structured in ways that can make the landmark based map representation difficult to implement. A second required assumption in the previously presented algorithms is that the noise on the observation sensors must be Gaussian. In reality this is almost never the case, even though as seen in the previous algorithms quality results can be achieved when making this assumption. In order to overcome these downfalls, in this chapter we augment the FastSLAM algorithm, introduced in Chapter 6, to replace the landmark based map with an occupancy grid.

The remainder of this chapter is organized as follows. In Section 7.2 occupancy grid

maps are introduced. In Section 7.3 the extension of the FastSLAM algorithm that makes use of occupancy grids is presented. Experimental results using the presented algorithm and our selected sensor suite are presented in Section 7.4 and concluding remarks are given in Section 7.5.

7.2. Occupancy Grid Maps

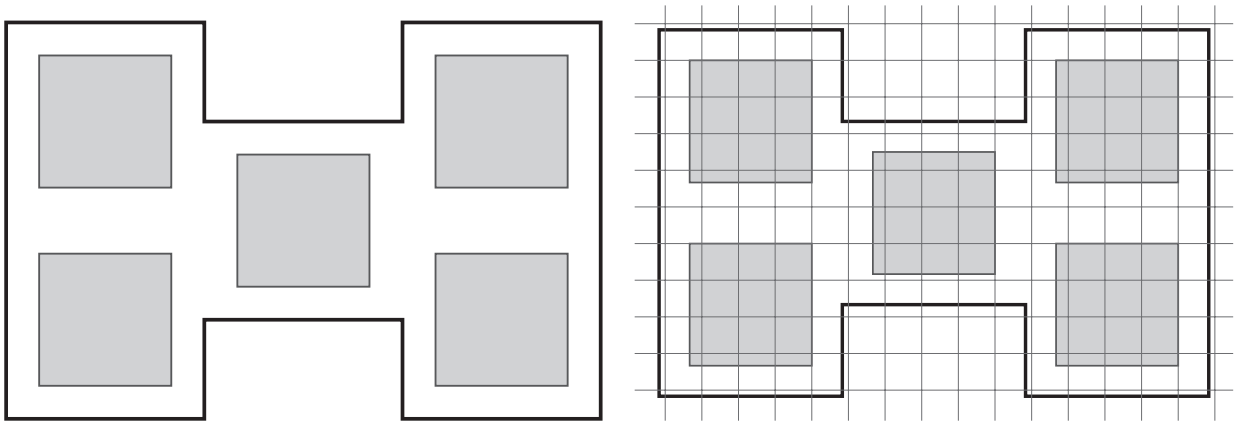
In the previous two chapters our sensor suite was validated using two *classic* SLAM algorithms, EKF SLAM and FastSLAM. The map generated by each of these algorithms represents the world using a set of landmarks, this type of map is referred to as a *geometric* map. Geometric maps suffer from several shortcomings when related to real-world applications for UMVs. First, in order to generate a geometric map it is assumed that the environment in which a UMV operates is structured in such a way that the sensors attached to the UMV can extract features that correspond to landmarks. In many situations this is a valid assumption, for example a UMV operating in an indoor office environment could use doorways, walls, corners, or furniture as landmarks to represent the environment. There are however many situations in which UMVs must operate in unstructured environments, such as a UMV operating in a disaster zone performing search an rescue missions. In these types of environments the use of geometric maps is difficult, if not impossible, due to the unstructured nature of the environment. The second downfall that geometric maps suffer from is their use adds an additional computational step. The sensors used by UMVs to see the world do not typically return feature locations, rather they provide an overall view of the environment that surrounds the UMV. Because of this the data generated by the sensor must be processed in order to extract the locations of features in the sensor data. This adds complexity as a method of extracting valid features must be developed and it also degrades the performance of the algorithm, even if slightly so, by adding an additional computing task

that must be performed. The process of extracting features from sensor data leads to the third issue with the use of geometric maps. During the extraction of features from the sensor data a significant amount of the information about the environment provided by the sensor data can be lost. As an example consider a laser range finder, a common sensor used on UMVs. A typical set of measurements generated by the sensor is a set of ranges over a 180° arc in front of the sensor at 1° increments. When this measurement is used in a geometric map, a single scan might contain four or five features in the data. By using a geometric map only the small subset of sensor data that represents those features is used to improve the estimate. However, all of the measurements returned by the sensor tell us something about the environment, even max range measurements that represents areas where nothing is seen provide us with information about the environment. The final disadvantage of geometric maps is related to how the map of the environment is typically used by a UMV. In many situations the map generated by the SLAM algorithm will be used by the UMV to complete more high level tasks, for example path planning with obstacle avoidance. In this situation by representing the environment using the location of landmarks in the environment successfully completing these high level tasks can be difficult. In the case of path planning with obstacle avoidance it is desirable to generate a path through the environment that avoids obstacles, however, if just the location of the corners of objects are stored in the map this task can be difficult to complete as additional information is required such as the walls or object edges that connect the corners.

In order to overcome the disadvantages described above we will be augmenting the FastSLAM algorithm to store the map of the environment using occupancy grids. Occupancy grids belong to a second family of map representations referred to as *spatial* maps. Occupancy grids were developed in [78, 79] and attempt to calculate the posterior of the map,

$$p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right), \tag{7.1}$$

based on the trajectory of the UMV, $\boldsymbol{\xi}_{1:k}$ where $\boldsymbol{\xi}_i \in \mathbb{R}^3$, $i = 1, \ldots, k$, and the history of sensor measurements, $\mathbf{z}_{1:k}$ where $\mathbf{z}_i \in \mathbb{R}^m$, $i = 1, \ldots, k$. Occupancy grids decompose the environment in which a UMV operates into a set of discrete cells by overlaying the environment with a spatial lattice. An example of this spacial decomposition is shown in Figure 7.1 where Figure 7.1a shows a simulated environment in which a UMV operates and Figure 7.1b shows the spacial lattice overlaying the environment used to decompose the world into a set of discrete cells.



(a) (b)

Figure 7.1: Example of an unknown environment in which an unmanned vehicle operates (a) the grid that is overlaid on the environment to represent the world (b).

By decomposing environment into a set of cells the occupancy grid used to represent an environment is stored using a matrix thus, $\mathbf{M} \in \mathbb{R}^{h \times w}$ where $h \in \mathbb{N}$ is the number of rows contained in the grid and $w \in \mathbb{N}$ is the number of columns. Each cell in the occupancy grid stores the probability that the location in the environment represented by that cell is occupied by an obstacle, thus the occupancy grid of an environment is given as

$$\mathbf{M} = \{\mathbf{M}(i, j) \in [0, 1]\}, \tag{7.2}$$

where $\mathbf{M}(i, j) = 0$ represents locations in the environment which are not filled with obstacles and referred to as *free* space and $\mathbf{M}(i, j) = 1$ is the opposite scenario in which the area of

the environment represented by the cell at (i, j) has an obstacle contained it and these areas are referred to as *occupied* space. An advantage of breaking down the environment into a set of discrete is that if we assume that the probability of occupancy contained in each cell of the occupancy grid is independent then (7.1) can be approximated as the product of cellular posteriors which is given as

$$p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right) = \prod_{i,j} p\left(\mathbf{M}\left(i, j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right), \tag{7.3}$$

where $p\left(\mathbf{M}\left(i, j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ is the probability that the cell located at (i, j) is occupied based on the vehicle trajectory $\boldsymbol{\xi}_{1:k}$ and history of sensor measurements $\mathbf{z}_{1:k}$. Each of the cellular posteriors is a binary estimation problem with static state as areas of the environment can be either occupied space or free space and we assume that the environment is static. A solution to problems of this form is the binary Bayes filter presented in [53]. In order to prevent truncation errors that can occur near the limits of $[0, 1]$ the binary Bayes filter stores $p\left(\mathbf{M}\left(i, j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ using log odds form. The log odds form of $x \in \mathbb{R}$ is given as

$$l(x) \triangleq \log \frac{x}{1 - x}, \tag{7.4}$$

and x can be recovered from the log odds form according to

$$x \triangleq 1 - \frac{1}{1 + \exp(l(x))}. \tag{7.5}$$

In occupancy grid mapping each time that a sensor measurement provides information about the probability of occupancy of a given cell the value of that cell is updated using the new information. As a result the probability of occupancy of locations that have been observed through time step k, $p\left(\mathbf{M}_k\left(i, j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$, quickly approaches either 0 for locations free of obstacles or 1 for cells that are occupied. While the addition of new information makes $p\left(\mathbf{M}_k\left(i, j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ quickly approach the limits it should never actually equal 0 or 1 as the would assume that the probability of occupancy is known perfectly, in reality this is not

the case as we are observing the world using imperfect sensors that are corrupted by noise. Depending on the data type selected to store $p\left(\mathbf{M}_k\left(i,j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ the repetitive inclusion of new information can quickly exceed the precision required to store the probability, thus $p\left(\mathbf{M}_k\left(i,j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ will be rounded to either 0 or 1. Once the probability of occupancy has been rounded to the limits any new information about occupancy of the cell is lost. To overcome this problem the log odds of $p\left(\mathbf{M}_k\left(i,j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ is stored by the binary Bayes filter. By storing the log odds form of the probability of occupancy the small range of potential values $[0, 1]$ are spread over a much larger range $(-\infty, \infty)$. By spreading the small range of potential value over the larger range the data type used to store the probability of occupancy has more precision to store $p\left(\mathbf{M}_k\left(i,j\right) \mid \boldsymbol{\xi}_{1:k}, \mathbf{z}_{1:k}\right)$ thus preventing the truncation/rounding errors from occurring. The occupancy grid update algorithm based on the binary Bayes filter can be seen in Algorithm 7.1.

Algorithm 7.1 Overview of the standard occupancy grid update algorithm.

procedure OGUPDATE($\boldsymbol{\xi}_k$,$\mathbf{M}_{k-1}$, $\mathbf{z}_k$)
    Data:
        $\boldsymbol{\xi}_k$ the current UMV pose
        $\mathbf{M}_{k-1}$ the occupancy grid from the previous time step, of size $w \times h$
        $\mathbf{z}_k$ the current sensor measurement, of length m

    Result:
        $\mathbf{M}_k$ the occupancy grid for the current time step

    for $i = 1, h$ **do**
        for $j = 1, w$ **do**
            for $l = 1, m$ **do**
                if $\mathbf{M}_{k-1}\left(i,j\right)$ lies in the perceptual range of $\mathbf{z}_k(l)$ **then**
                    $\mathbf{M}_k\left(i,j\right) \leftarrow \mathbf{M}_{k-1}\left(i,j\right) + \log \frac{p(\mathbf{M}(i,j) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l))}{1-p(\mathbf{M}(i,j) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l))} - l_0$
                end if
            end for
        end for
    end for
end procedure

In Algorithm 7.1 $p(\mathbf{M}(i,j) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l))$ is the inverse sensor model of the sensor being used by the UMV to observe the world. It returns the probability that a cell in the environment is occupied based on the current pose of the UMV, which is assumed to be known in occupancy grid mapping, and the current sensor measurement. The value l_0 is defined as

$$l_0 = \log \frac{p(\mathbf{M}(i,j))}{1 - p(\mathbf{M}(i,j))}, \tag{7.6}$$

and is referred to as the prior of occupancy. The prior of occupancy is the probability of occupancy for a cell before the inclusion of any sensor measurements are included and $p(\mathbf{M}(i,j))$ is typically selected to be 0.5.

7.3. FastSLAM with Occupancy Grids Algorithm

We will now extend the classic FastSLAM algorithm presented in Chapter 6 by replacing the map that stores the location of landmarks in the environment with occupancy grids. In a probabilistic sense the FastSLAM OG algorithm estimates

$$p(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}), \tag{7.7}$$

which is the distribution that represents the trajectory of a UMV $\boldsymbol{\xi}_{1:k}$ and the map of the environment $\mathbf{M}$ based on the series of control inputs $\mathbf{u}_{1:k}$ and observations $\mathbf{z}_{1:k}$. By estimating the distribution that represents the trajectory, (7.7) can be factored into a pair of simpler terms based on the property of conditional independence which is given by

$$p(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}) = p(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k})\, p(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}), \tag{7.8}$$

where $p(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k})$ is the distribution that represents the UMV trajectory and the distribution that represents the map of the environment is denoted $p(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k})$. The FastSLAM OG approach estimates the factored SLAM posterior (7.8) using a Rao-Blackwellized particle filter. The Rao-Blackwellized particle filter is a version of a sampling

importance resampling (SIR) ([76]) particle filter and it is the SIR particle filter that provides the basis of the FastSLAM OG solution. The Rao-Blackwellized particle filter estimates a portion of the state, in FastSLAM OG the UMV trajectory $\boldsymbol{\xi}_{1:k}$ using a particle filter, and each particle in the particle filter maintains its own estimate of the map $\mathbf{M}$. As a result, each particle in the particle filter is composed of a UMV pose and a map, in this approach an occupancy grid, thus the pth particle at time k is defined as

$$\mathcal{X}_k^{[p]} \triangleq \left[\begin{array}{cc} \boldsymbol{\xi}_k^{[p]} & \mathbf{M}_k^{[p]} \end{array} \right], \tag{7.9}$$

where $\boldsymbol{\xi}_k^{[p]}$ is the pose estimate of the pth particle at time step k and $\mathbf{M}_k^{[p]}$ is the occupancy grid of the environment maintained by the particle at time step k. The FastSLAM OG approach is implemented in a four-step procedure based on using a SIR particle filter to estimate (7.7) and an overview of the algorithm is presented in Algorithm 7.2 for time step k.

7.3.1. Pose Sampling

In the first step of the algorithm (Line 12 in Algorithm 7.2) a new particle pose is generated and added to a set of temporary particles, $\overline{\mathcal{X}}_k$. The new pose for the pth particle is generated by sampling from the probabilistic motion model of the UMV

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right). \tag{7.10}$$

The probabilistic motion model of the UMV takes the pose of a particle in $\mathcal{X}_{k-1}$ and generates a new pose using the state transition model of the UMV, the previous pose estimate of the particle $\boldsymbol{\xi}_{k-1}^{[p]}$, the current control input $\mathbf{u}_k$, and the characteristics of the noise on $\mathbf{u}_k$. The result is that the particles in $\overline{\mathcal{X}}_k$ are distributed according to

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right), \tag{7.11}$$

Algorithm 7.2 Overview of the FastSLAM OG algorithm

procedure FastSLAMOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
    Data:
        $\mathcal{X}_{k-1}$ the particle set from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of measurements

    Result:
        $\mathcal{X}_k$ the particle set from the current time step

    $\overline{\mathcal{X}}_k \leftarrow \{\}$
    for $p \leftarrow 1, n$ **do**
        $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
        $w^{[p]} \leftarrow \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$
        $\mathbf{M}_k^{[p]} = \text{OGUpdate}(\boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}, \mathbf{z}_k)$
        $\overline{\mathcal{X}}_k^{[p]} \leftarrow \begin{bmatrix} \boldsymbol{\xi}_k^{[p]} & \mathbf{M}_k^{[p]} & w^{[p]} \end{bmatrix}$
    end for
    $\mathcal{X}_k \leftarrow \{\}$
    for $p = 1, n$ **do**
        select j with probability proportional to $w^{[p]}$
        $\mathcal{X}_k^{[p]} \leftarrow \overline{\mathcal{X}}_k^{[j]}$
    end for
end procedure

which is referred to as the *proposal distribution*. The actual distribution which we are attempting to estimate is referred to as the *target distribution* and given as

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right). \tag{7.12}$$

The proposal distribution is transformed to the target distribution in the last stage of the algorithm, the resampling stage (Lines 18-21 in Algorithm 7.2). The details of the pose sampling procedure used can be seen in Section C.1.1. Before the resampling process can occur, $\mathbf{z}_k$ must be incorporated into the estimate, which is achieved in the calculation of the particle's importance weight.

7.3.2. Importance Weight Calculation

In the second step of the algorithm an importance weight for each particle is generated. The need for an importance weight for each particle comes from our use of a SIR particle filter and, as seen in the previous section, we are sampling particles from a distribution that does not match the distribution that we are attempting to estimate. From [77] if we are unable to directly sample from the distribution that we are attempting to estimate, then if the importance weight for each particle is defined as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \tag{7.13}$$

and particles are drawn from $\overline{\mathcal{X}}_k$ and added to $\mathcal{X}_k$ with a probability proportional to $w_k^{[p]}$ then $\mathcal{X}_k$ will approximate the target distribution and the quality of the approximation will improve as the number of particles increases.

Using (7.13) along with the previously presented distributions (7.11) and (7.12) the importance weight of each particle simplifies to

$$w_k^{[p]} = \frac{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)}{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right)} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right), \tag{7.14}$$

where $p\left(\mathbf{z}_k \mid \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$ is the probabilistic measurement model of the sensor being used by the UMV and $\eta > 0$ is a normalizing factor. The probabilistic measurement model is dependent on the perceptual sensor being used by the UMV and it calculates the probability that the sensor measurement would be made based on the current pose estimate and map currently maintained by the particle. This form of importance weight incorporates the sensor observations into the estimate that were not included in the pose sampling procedure. The details of the importance weight calculation can be found in Section C.2. Once the importance wight has been calculated for each particle then the occupancy grid maintained by each particle is updated using the new particle pose and current sensor measurement.

7.3.3. Occupancy Grid Update

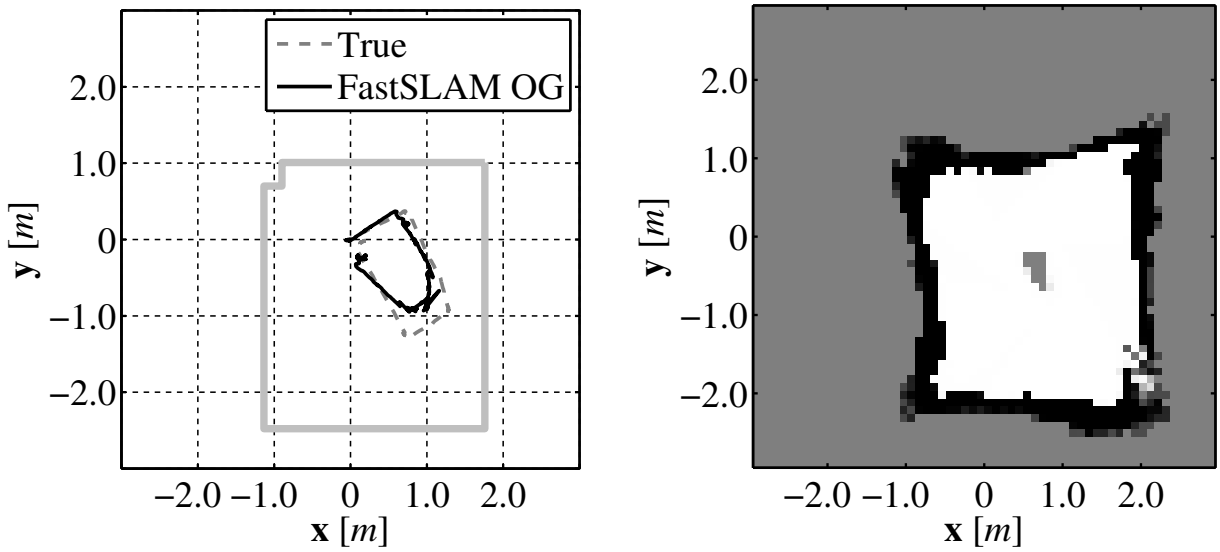
The next step in the FastSLAMOG algorithm is to update the occupancy grid of the particle. The occupancy grid of each particle, $\mathbf{M}_k^{[p]}$, is updated using the occupancy grid update algorithm discussed Section 7.2 with an overview of the algorithm presented in Algorithm 7.1. Further details of the occupancy grid update can be found in Section C.3.

7.3.4. Resampling

After the occupancy grid for each particle in $\overline{\mathcal{X}}_k$ has been updated, the final step of the algorithm is the resampling process. During the resampling process particles are drawn with replacement from $\overline{\mathcal{X}}_k$ with a probability proportional to $w^{[p]}$. This resampling step takes the set of temporary particles that are distributed according to (7.11) and ensures that there is a high probability that $\mathcal{X}_k$ is distributed according to (7.12). For the experimental results presented in the following section the algorithm used to perform the resampling is the same used in the previous chapter and the details of the algorithm can be seen in Appendix B Section B.5.

7.4. Experimental Results

To test the FastSLAM OG algorithm the same test platform and test environment used to test the previous algorithms was used. The final path estimate and map generated by the algorithm can be seen in Figure 7.2 and the error in the location estimate over time is shown in Figure 7.3. From the results it can be seen that the largest position error is < 0.4m. To illustrate the incremental update procedure of the algorithm a series of equally spaced frames of the position estimate are presented in Figure 7.4 and the occupancy grid at the corresponding frames can be seen in Figure 7.5. To generate the position plot the mean position estimate of the particle set is used and the uncertainty ellipse is generated using the covariance of the particle set. The corresponding occupancy grid that is displayed is the occupancy grid maintained by the particle with largest importance weight. It can be seen that the uncertainty in the position estimate remains much smaller than that of the previously discussed algorithms.



(a) (b)

Figure 7.2: The FastSLAM OG produced path estimate (a) and map (b).

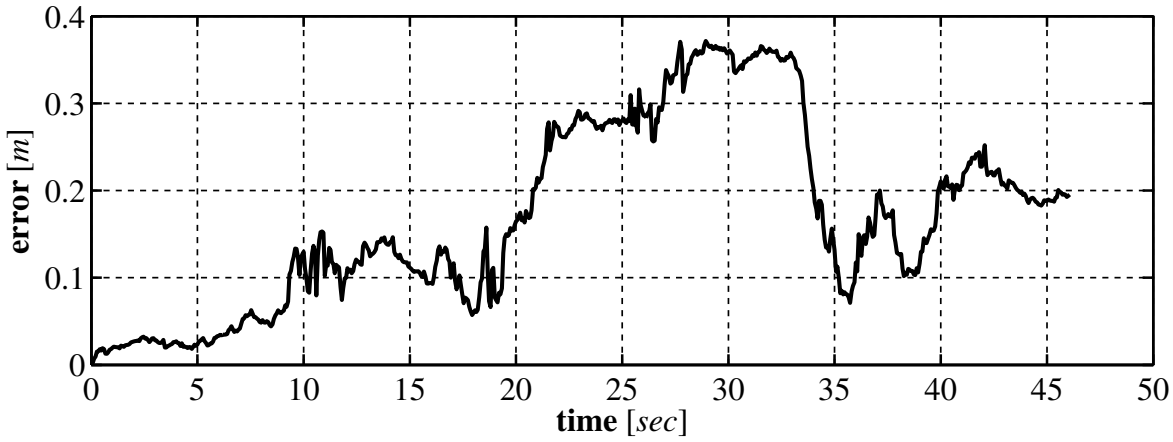


Figure 7.3: The error in the FastSLAM OG position estimate over time.

To illustrate the advantage of using the FastSLAM OG algorithm over the raw sensor measurements Figure 7.6 shows a comparison of the position error between the FastSLAM OG algorithm and the estimate produced using the raw odometry data. Finally, Figure 7.7 shows a comparison of the *true* occupancy grid generated using the baseline position estimate and a LiDAR sensor compared with that generated by the FastSLAM OG algorithm and that generated using the position estimate generated using the raw odometry measurements. The raw odometry map has a mean squared error (MSE) when compared to the true map of 0.0684 while the map generated by the FastSLAM OG algorithm has a MSE of 0.0462.

7.5. Conclusions

In this chapter an algorithm was introduced that estimates the position of the robot and generates a map of the environment using an occupancy grid. The use of occupancy grids to represent the environment as opposed to discrete landmarks provides the advantage that there is no assumption placed on the structure of the environment and a more detailed map of the environment is constructed. The algorithm makes no assumption on the noise affecting the noise on the control input or the noise on the sensor observations. There is

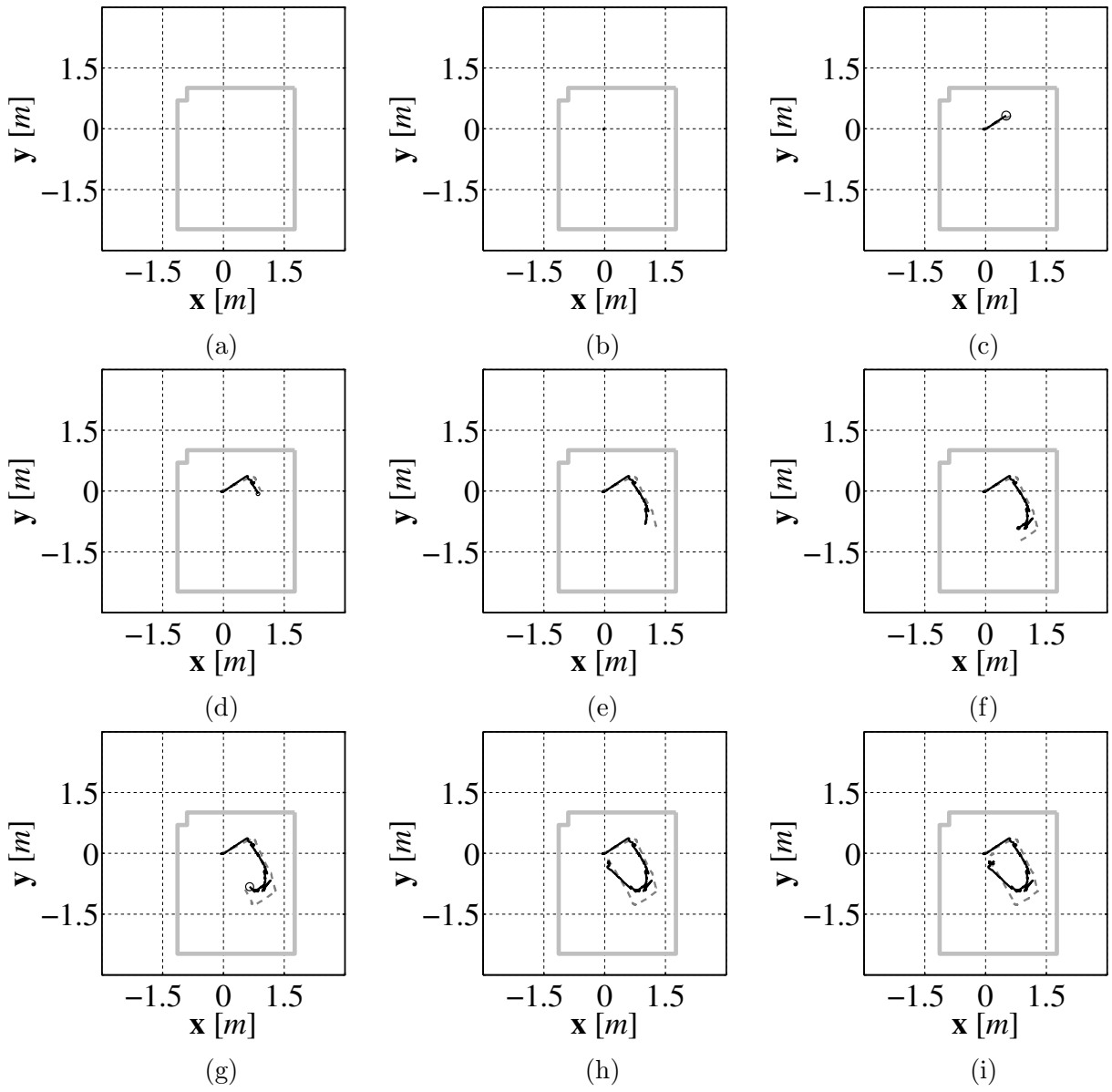


Figure 7.4: Evenly spaced sequence of path estimates produced by the FastSLAM OG algorithm.

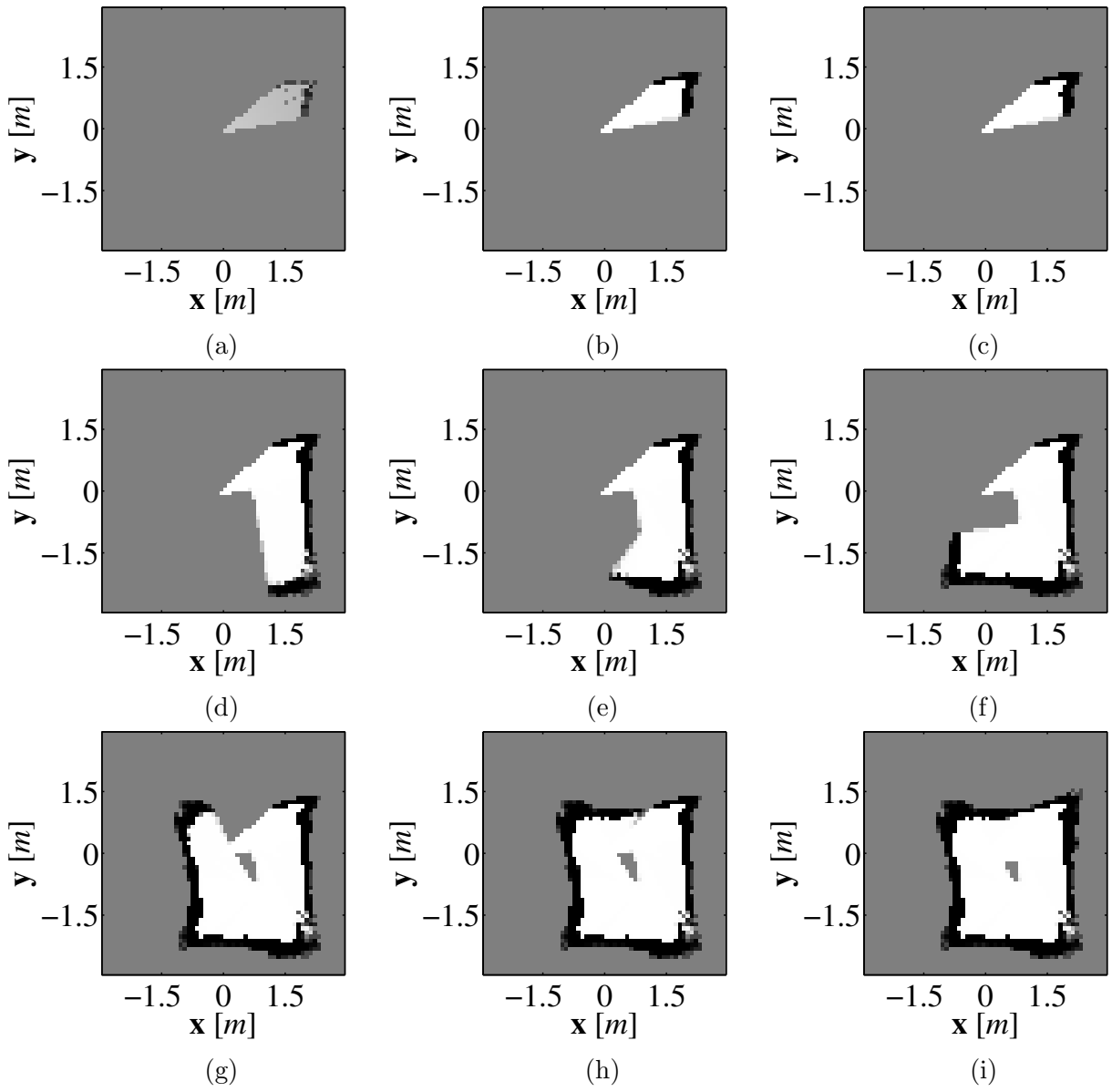


Figure 7.5: Evenly spaced sequence of map estimates produced by the FastSLAM OG algorithm.

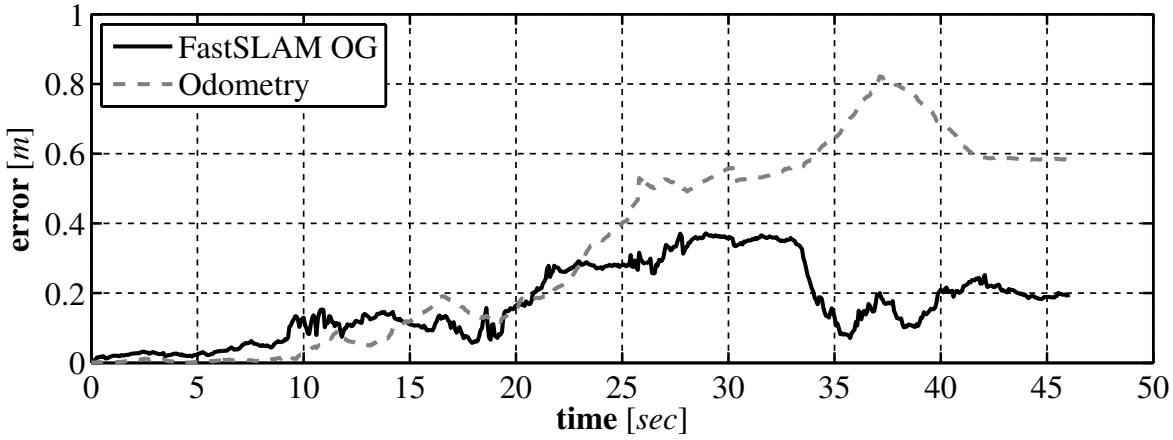


Figure 7.6: Comparison of the FastSLAM OG position error to the VO position error.

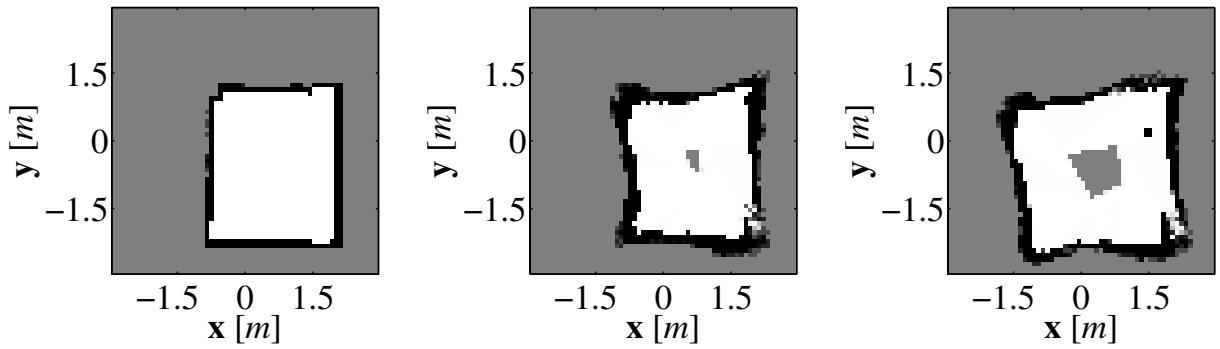


Figure 7.7: Comparison of the true occupancy grid (a), the FastSLAM OG occupancy grid (b), and VO occupancy grid (c).

downside in using occupancy grids to represent the environment in a particle filter based SLAM algorithm. Depending on the size of the environment in which the UMV operates and the amount of detail about the environment that is required, the size of the occupancy grid maintained by each particle can become quite large, containing thousands or hundreds of thousands of elements. As discussed in Section 7.3.2 the quality of the estimate increases as the number of particles in the particle set increases, thus, it can be easily seen that a large amount of computer memory can be required perform the algorithm. This concern is the motivating factor for the remainder of the research presented in this dissertation and the first step in solving this problem is presented in the following chapter.

Chapter 8

Wavelet Representation of Occupancy Grids For FastSLAM Implementation

8.1. Introduction

In the previous chapter a version of the FastSLAM algorithm was presented that uses occupancy grids to represent the map of the environment in which a UMV operates. The FastSLAM OG algorithm has many advantages over the classic FastSLAM algorithm due to the use of occupancy grids. The map generated by the FastSLAM OG algorithm gives a much fuller picture of the environment in which the UMV operates, and it allows for all of the data generated by the sensor that observes the environment to be used to generate the map. Occupancy grids also serve as the basis for many robotic algorithms that have already been developed, for example path planning with obstacle avoidance ([80]). As discussed in the previous chapter the estimate generated by the FastSLAM OG algorithm improves as the number of particles in the particle filter increases. However, this can cause an issues as increasing the number of particles in the particle filter quickly increases the amount of computer memory required by the algorithm due to each particle storing an occupancy grid of the environment. To overcome this problem we wish to develop an algorithm that reduces the amount of memory required by storing the occupancy grid in compressed form.

In this chapter an extension of the FastSLAM OG algorithm is presented that stores the occupancy grid maintained by each particle using an alternate basis (FastSLAM AOG). The development of the FastSLAM AOG algorithm is required before the occupancy grid compression can be implemented in the following chapter. The FastSLAM AOG algorithm described in this chapter extends the FastSLAM OG algorithm presented in Chapter 7 so that the occupancy grid maintained by each particle in stored using an alternate basis, in particular the Haar wavelet basis.

The remainder of this chapter is organized as follows. In Section 8.2 the extension of the FastSLAM OG algorithm that stores the occupancy grid using an alternate basis is presented. An overview of the Haar wavelet representation, the alternate basis that is used in our experimental results is presented in Section 8.3 with experimental results presented in Section 8.4. Finally, some concluding remarks are presented in Section 8.5.

8.2. FastSLAM With Occupancy Grids Stored Using an Alternate Basis

As discussed in previous chapters the goal of SLAM is to estimate the pose $\boldsymbol{\xi}_k$ of a UMV and map $\mathbf{M}$ of the unknown environment in which the UMV is operating at some time $k > 0$. As opposed to many SLAM solutions that attempt to solve the online SLAM problem, that is estimating the posterior that represents the pose of a UMV and map of the environment based on a set of control inputs and sensor measurements, the FastSLAM approach attempts to estimate the full SLAM posterior. The full SLAM posterior is given as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{8.1}$$

which is the distribution that represents the trajectory of a UMV $\boldsymbol{\xi}_{1:k}$ and the map of the environment $\mathbf{M}$ based on the series of control inputs $\mathbf{u}_{1:k}$ and observations $\mathbf{z}_{1:k}$. By

estimating the full SLAM posterior (8.1) can be factored, using the property of conditional independence, into a pair of simpler terms which is given by

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{8.2}$$

where $p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)$ is the distribution that represents the UMV trajectory and the distribution that represents the map is denoted $p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)$. The FastSLAM AOG approach estimates the factored SLAM posterior (8.2) using a Rao-Blackwellized particle filter. The Rao-Blackwellized particle filter is a version of a sampling importance resampling (SIR) ([76]) particle filter and it is the SIR particle filter that provides the format of the FastSLAM solution. The Rao-Blackwellized particle filter estimates the distribution representing a portion of the state, which in FastSLAM AOG is the UMV trajectory $\boldsymbol{\xi}_{1:k}$, using a particle filter, and each particle in the particle filter maintains its own estimate of the remainder of the state, which is the map $\mathbf{M}$. The FastSLAM AOG algorithm stores the occupancy grid maintained by each particle in an alternate basis and the details of that representation are presented in the following section. The FastSLAM AOG algorithm is implemented in a four-step procedure and an overview of the algorithm is shown in Algorithm 8.1.

8.2.1. Alternate Occupancy Grid Representation

Before discussing the FastSLAM AOG algorithm in detail, we will first address how $\mathbf{M}_k^{[p]}$ is stored using some alternate basis. Before changing the representation, the occupancy grid $\mathbf{M}_k^{[p]}$, which is a $w \times h$ matrix, is converted to vector form, $\mathbf{m}_k^{[p]} \in \mathbb{R}^n$ where $n = wh$ and we refer to the vector form of the occupancy grid as $\mathbf{m}_k^{[p]} \triangleq \text{vec}\left(\mathbf{M}_k^{[p]}\right)$. An element in the occupancy grid $\mathbf{M}_k^{[p]}(i,j)$ is accessed in vector form as $\mathbf{m}_k^{[p]}(l)$ where $l = iw + j$. The vector form of the occupancy grid can then be represented using some alternate basis as

$$\mathbf{c}_k^{[p]} = \boldsymbol{\Phi}^{-1}\mathbf{m}_k^{[p]}, \tag{8.3}$$

Algorithm 8.1 Overview of the FastSLAM AOG algorithm

procedure FastSLAMAOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
    Data:
        $\mathcal{X}_{k-1}$ the particle set from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of measurements

    Result:
        $\mathcal{X}_k$ the particle set from the current time step

    $\overline{\mathcal{X}}_k \leftarrow \{\}$
    for $p = 1, n$ **do**
        $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
        $w^{[p]} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \boldsymbol{\Phi}\mathbf{c}_{k-1}^{[p]}\right)$
        $m = \text{LENGTH}(\mathbf{c}_{k-1}^{[p]})$
        $\boldsymbol{\alpha}_k^{[p]} = \text{AlphaConstruction}(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k, m)$
        $\mathbf{c}_k^{[p]} = \mathbf{c}_{k-1}^{[p]} + \boldsymbol{\Phi}^{\mathrm{T}}\left[\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta}\right]$
        $\overline{\mathcal{X}}_k \leftarrow \left[\begin{array}{ccc} \boldsymbol{\xi}_k^{[p]} & \mathbf{c}_k^{[p]} & w^{[p]} \end{array}\right]$
    end for
    $\mathcal{X}_k \leftarrow \{\}$
    for $j = 1, n$ **do**
        draw p with probability proportional to $w^{[p]}$
        $\mathcal{X}_k^{[j]} \leftarrow \overline{\mathcal{X}}_k^{[p]}$
    end for
end procedure

where $\mathbf{c}_k^{[p]} \in \mathbb{R}^n$ is a set of coefficients which represent the occupancy grid in the alternate basis and $\mathbf{\Phi} \in \mathbb{R}^{n \times n}$ is a transformation matrix that describes the relationship between $\mathbf{c}_k^{[p]}$ and $\mathbf{m}_k^{[p]}$. If the transformation matrix is orthonormal, which is valid for many commonly used transformations, then $\mathbf{\Phi}^{-1}$ in (8.3) can be replaced with $\mathbf{\Phi}^{\mathrm{T}}$.

The FastSLAM AOG algorithm is a modification of Algorithm 7.2 presented in Chapter 7 where each particle stores the set of coefficients that represent the occupancy grid $\mathbf{c}_k^{[p]}$ as opposed to the full occupancy grid $\mathbf{M}_k^{[p]}$. Using this representation each particle in $\mathcal{X}_k$ is defined as

$$\mathcal{X}_k^{[p]} \triangleq \begin{bmatrix} \boldsymbol{\xi}_k^{[p]} & \mathbf{c}_k^{[p]} \end{bmatrix}, \tag{8.4}$$

where $\boldsymbol{\xi}_k^{[p]}$ is the pose estimate of the pth particle at time step k and $\mathbf{c}_k^{[p]}$ is the set of coefficients that represent the occupancy grid that has been generated through k. In the first step of the algorithm a new set of particles are generated by sampling from the probabilistic motion model of the UMV and that step is discussed in the following section.

8.2.2. Pose Sampling

As with the previously described implementations of the FastSLAM algorithm (Chapter 6 and Chapter 7), in the first step of the FastSLAM AOG algorithm (Line 12 in Algorithm 8.1) a set of new particle poses are generated and added to a set of temporary particles, $\overline{\mathcal{X}}_k$. The new pose for the pth particle is generated by sampling from the probabilistic motion model of the UMV

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right). \tag{8.5}$$

The probabilistic motion model of the UMV generates a new pose based on the state transition model of the UMV, the pose of the pth particle at time step $k-1$, the current control input $\mathbf{u}_k$, and the characteristics of the noise on $\mathbf{u}_k$. The pose sampling process used in the experimental results to follow is the same approach used in the previous two algorithms and

the details can be seen in Appendix B Section B.2. The result is that the particles in $\overline{\mathcal{X}}_k$, generated by the pose sampling process, are distributed according to

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right), \tag{8.6}$$

which is referred to as the *proposal distribution*. The actual distribution which we are attempting to estimate is referred to as the *target distribution*, the first half of (8.2), and given as

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right). \tag{8.7}$$

The proposal distribution is transformed to the target distribution in the last stage of the algorithm, the resampling stage (Lines 20-23 in Algorithm 8.1). Before the resampling process can occur, $\mathbf{z}_k$ must be incorporated into the estimate, which is achieved in the calculation of the particle's importance weight.

8.2.3. Importance Weight Calculation

In the second step of the algorithm an importance weight for each particle is generated. The need for an importance weight for each particle comes from our use of a SIR particle filter and as seen in the previous section we are sampling particles from a distribution that does not match the distribution that we are attempting to estimate. From [77] if we are unable to directly sample from the distribution that we are attempting to estimate, then if the importance weight for each particle is given as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \tag{8.8}$$

and particles are drawn from $\overline{\mathcal{X}}_k$ with replacement and added to $\mathcal{X}_k$ with a probability proportional to $w_k^{[p]}$ then $\mathcal{X}_k$ will approximate the target distribution and the quality of the approximation will improve as the number of particles increases.

Using (8.8) along with the previously discussed distributions, (8.7) and (8.6), the importance weight of each particle simplifies to ([81])

$$w_k^{[p]} = \frac{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)}{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right)} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right), \tag{8.9}$$

where $p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$ is the probabilistic measurement model of the sensor being used by the UMV and $\eta > 0$ is a normalizing factor. The probabilistic measurement model is dependent on the perceptual sensor being used by the UMV and it calculates the probability that the sensor measurement would be made based on the current pose estimate and map maintained by the particle. The probabilistic measurement model that we use is the beam model ([53]) and the details can be seen in Section C.2. In a simplistic form the beam model first generates an expected sensor measurement based on $\boldsymbol{\xi}_k^{[p]}$ and $\mathbf{M}_{k-1}^{[p]}$. The actual measurement $\mathbf{z}_k$ is then compared to the expected measurement to determine the probability that $\mathbf{z}_k$ would be made based on $\boldsymbol{\xi}_k^{[p]}$ and $\mathbf{M}_{k-1}^{[p]}$. In order to generated the expected value, the measurement model must have access to the probability of occupancy for each cell in the grid in order to determine what cells represent objects that would be measured by the sensor. When storing the occupancy grid using the alternate basis the grid passed to the probabilistic measurement model must contain the probability of occupancy for each cell. Knowing this, (8.9) is restated as

$$w_k^{[p]} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \boldsymbol{\Phi}\mathbf{c}_{k-1}^{[p]}\right) \tag{8.10}$$

where $\boldsymbol{\Phi}\mathbf{c}_{k-1}^{[p]}$ converts the set of coefficients to the set of probability of occupancy values that are required to generate the expected sensor measurement. Once the current sensor measurement has been incorporated into the estimate the occupancy grid, stored by each particle using an alternate basis, must be updated to include the current sensor measurements.

8.2.4. Occupancy Grid Update

Once the important weight for a given particle has been generated, the occupancy grid for each particle must be updated before the resampling process. As seen in Section 7.2, the occupancy grid update is performed by updating all of the cells that fall within the perceptual range of the sensor being used to observe the environment using the binary Bayes filter from [53]. Using this approach the update of a single cell that falls within the perceptual range of a single observation is given by

$$\mathbf{M}_k^{[p]}(i,j) = \mathbf{M}_{k-1}^{[p]}(i,j) + \log \frac{p\left(\mathbf{M}(i,j) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k\right)}{1 - p\left(\mathbf{M}(i,j) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k\right)} - l_0, \tag{8.11}$$

where $p\left(\mathbf{M}(i,j) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k\right)$ is the inverse sensor model that describes the probability that a cell is occupied based on the pose and measurement, and l_0 is the prior of occupancy, defined in Chapter 7 Section 7.2, which provides the probability that a cell is occupied before any measurements are made.

Using this approach we write the occupancy grid update in vector form as

$$\mathbf{m}_k^{[p]} = \mathbf{m}_{k-1}^{[p]} + \boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta}, \tag{8.12}$$

where $\mathbf{m}_k^{[p]}$ is the updated occupancy grid in vector form, $\boldsymbol{\beta} \in \mathbb{R}^n$ is a vector where each element is equal to l_0 and $\boldsymbol{\alpha}_k^{[p]} \in \mathbb{R}^n$ is a vector that performs the occupancy grid update, the generation of this vector can be seen in Algorithm 8.2. Using the idea that the occupancy grid is represented using some alternate basis, (8.12) is rewritten using (8.3) as

$$\mathbf{c}_k^{[p]} = \mathbf{c}_{k-1}^{[p]} + \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\beta}, \tag{8.13}$$

where, assuming that the transformation is orthonormal, $\boldsymbol{\Phi}^{\mathrm{T}}$ is used as opposed to $\boldsymbol{\Phi}^{-1}$, to convert the updating terms $\boldsymbol{\alpha}_k^{[p]}$ and $\boldsymbol{\beta}$ to the alternate basis. Equation (8.13) can be simplified by grouping the two update terms to reduce the number of matrix-vector multiplications

that must be performed to yield the update equation for occupancy grids stored using an alternate basis that is given by

$$\mathbf{c}_k^{[p]} = \mathbf{c}_{k-1}^{[p]} + \mathbf{\Phi}^{\mathrm{T}} \left(\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta} \right). \tag{8.14}$$

Once the occupancy grid, stored using an alternate basis, is updated for each particle the final step of the FastSLAM AOG algorithm is the resampling process.

Algorithm 8.2 Construction of the occupancy grid update vector $\boldsymbol{\alpha}_k^{[p]}$.

procedure AlphaConstruction($\boldsymbol{\xi}_k^{[p]}$, $\mathbf{z}_k$, n)
    Data:
        $\boldsymbol{\xi}_k^{[p]}$ the pose estimate of the current particle
        $\mathbf{z}_k$ the current set of measurements, of length m
        n the size of the vector form of the occupancy grid

    Result:
        $\boldsymbol{\alpha}_k^{[p]}$ the vector containing the occupancy grid update components

    for $i = 1, n$ **do**
        $\boldsymbol{\alpha}(i) = l_0$
    end for
    for $i = 1, n$ **do**
        for $j = 1, m$ **do**
            if i corresponds to a cell that lies in the perceptual range of $\mathbf{z}(j)$ **then**
                $\boldsymbol{\alpha}(i) = \log \frac{p\left(\mathbf{m}(i) \,\middle|\, \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(j)\right)}{1 - p\left(\mathbf{m}(i) \,\middle|\, \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(j)\right)}$
            end if
        end for
    end for
end procedure

8.2.5. Resampling

After the occupancy grid for each particle in $\overline{\mathcal{X}}_k$ has been updated, the final step of the algorithm is the resampling process. During the resampling process particles are drawn with replacement from $\overline{\mathcal{X}}_k$ with a probability proportional to $w^{[p]}$. This resampling step takes the

set of temporary particles that are distributed according to (8.6) and ensures that there is a high probability that $\mathcal{X}_k$ is distributed according to (8.7). The resampling procedure used in the experimental results presented later in this chapter is the same approach used in the previous two FastSLAM implementations and the details can be seen in Section B.5.

8.3. Haar Wavelet Basis

In the previous section the FastSLAM AOG algorithm was developed which represents the occupancy grid of each particle using some generic alternate basis. One of the simplest possible representations, which we will use in the experimental results, uses the Haar wavelet transformation matrix ([82]) and the corresponding set of coefficients to represent some discrete signal. In essence the Haar wavelet transform decomposes to a discrete signal into different resolutions. This means that when reconstructing a signal using its Haar transform coefficients, different resolution forms of the signal can be generated by selecting different coefficients. The details of the Haar wavelet transform will now be provided in the following section.

8.3.1. Haar Wavelet Transform

Following the discussion in [2] one of the key concepts which allows for a discrete signal to be decomposed into different resolutions is the idea of a vector space. A discrete signal that we are attempting to transform, that is a n dimensional vector of values, can be thought of as a piecewise constant function on $[0, 1)$. A discrete signal containing a single element is a constant value function over $[0, 1)$. The vector space V^0 contains all possible single element signals. The vector space V^1 contains all possible two element signals that are constant over $[0, 1/2)$ and $[1/2, 1)$. This idea can be generalized to say that the vector space V^j contains all possible functions that are piecewise-constant over 2^j subintervals of $[0, 1)$.

With the vector space V^j defined the next step, in following [2], is to define a basis for V^j. The basis used by the Haar wavelet transform for the vector space V^j is given as

$$\phi_i^j(x) \triangleq \phi\left(2^j x - i\right),\ i = 0, 1, \ldots, 2^j - 1, \tag{8.15}$$

where

$$\phi(x) \triangleq \begin{cases} 1 & \text{for } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{8.16}$$

The basis functions of V^j are known as *scaling* functions and are a scaled and translated box function. An example of the basis of V^1 is shown in Figure 8.1.

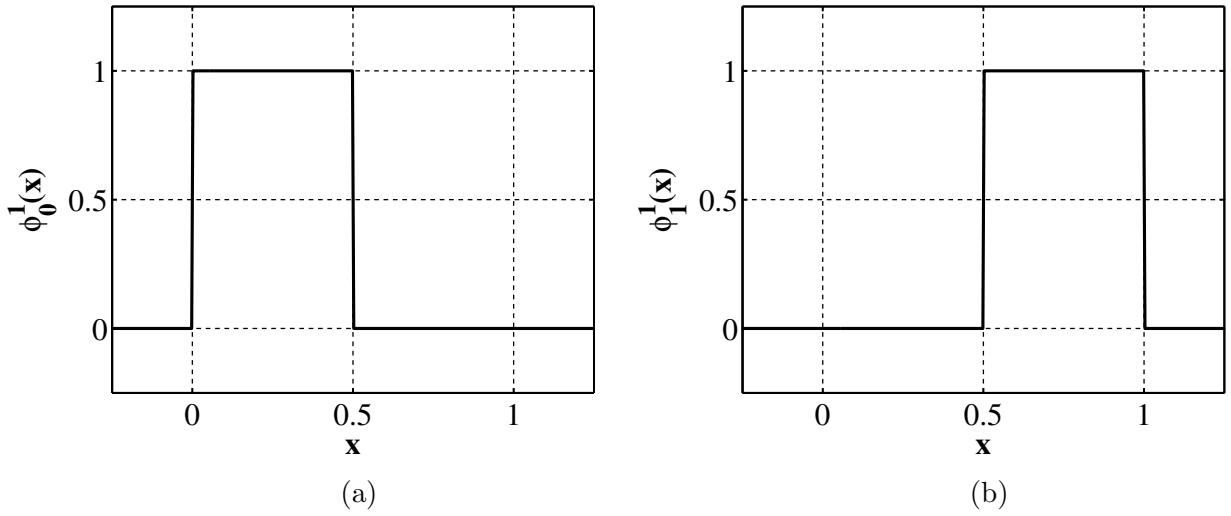


Figure 8.1: The scaling functions that serve as the basis for V^1.

After defining the basis of V^j a second vector space W^j is defined. The vector space W^j is the orthogonal complement of V^j in V^{j+1} according to the inner product defined as

$$\langle f(x) |\, g(x) \rangle \triangleq \int_0^1 f(x)\, g(x)\, dx, \tag{8.17}$$

with $f(\cdot), g(\cdot) \in V^j$. This means that W^j contains all functions in V^{j+1} that are orthogonal to all functions in V^j according to (8.17). In other words the vector space W^j and its basis are used to represent the parts of a function in V^{j+1} that cannot be represented in V^j ([2]).

The basis of W^j are known as wavelets and given by

$$\psi_i^j(x) \triangleq \psi\left(2^j x - i\right),\ i = 0, 1, \ldots, 2^j - 1, \tag{8.18}$$

where

$$\psi(x) \triangleq \begin{cases} 1 & \text{for } 0 \leq x < 1/2, \\ -1 & \text{for } 1/2 \leq x < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{8.19}$$

An example of the wavelet functions for W^2 can be seen in Figure 8.2. Using the coefficients that correspond to the scaling function of V^j and the wavelet functions of W^j any piecewise-constant function in V^{j+1} can be reproduced.

From [2], one of the key properties of the Haar wavelet transform is that the full set of basis functions used to represent a signal in V^{j+1} are orthogonal to each other, that is each of the scaling function in V^j and wavelet functions in W^j are orthogonal to each other according to (8.17). A second property of a given set of basis functions that is desirable in many cases is that the set of basis functions be normalized. A given basis function $f(x)$ is normalized if $\langle f(x) | f(x) \rangle = 1$. Normalization of the Harr basis functions can be achieved by replacing the scaling function (8.15) and wavelet function (8.18) with

$$\phi_i^j(x) \triangleq 2^{j/2}\phi\left(2^j x - i\right), \tag{8.20}$$

$$\psi_i^j(x) \triangleq 2^{j/2}\psi\left(2^j x - i\right). \tag{8.21}$$

8.3.2. Optimized Haar Wavelet Transform

In Section 8.2.1 it was shown that the occupancy grid could be transformed from the standard form to some alternate representation using a transformation matrix $\mathbf{\Phi}$ as described in (8.3). In general, the size of the occupancy grid can be quite large, for example in the experimental results presented in Section 8.4 the size of the occupancy grid is 64×64, hence when converted to vector form, the occupancy grid is in $\mathbb{R}^{4096}$. The resulting transformation matrix required to convert the occupancy grid to the Haar basis would be in $\mathbb{R}^{4096 \times 4096}$. A

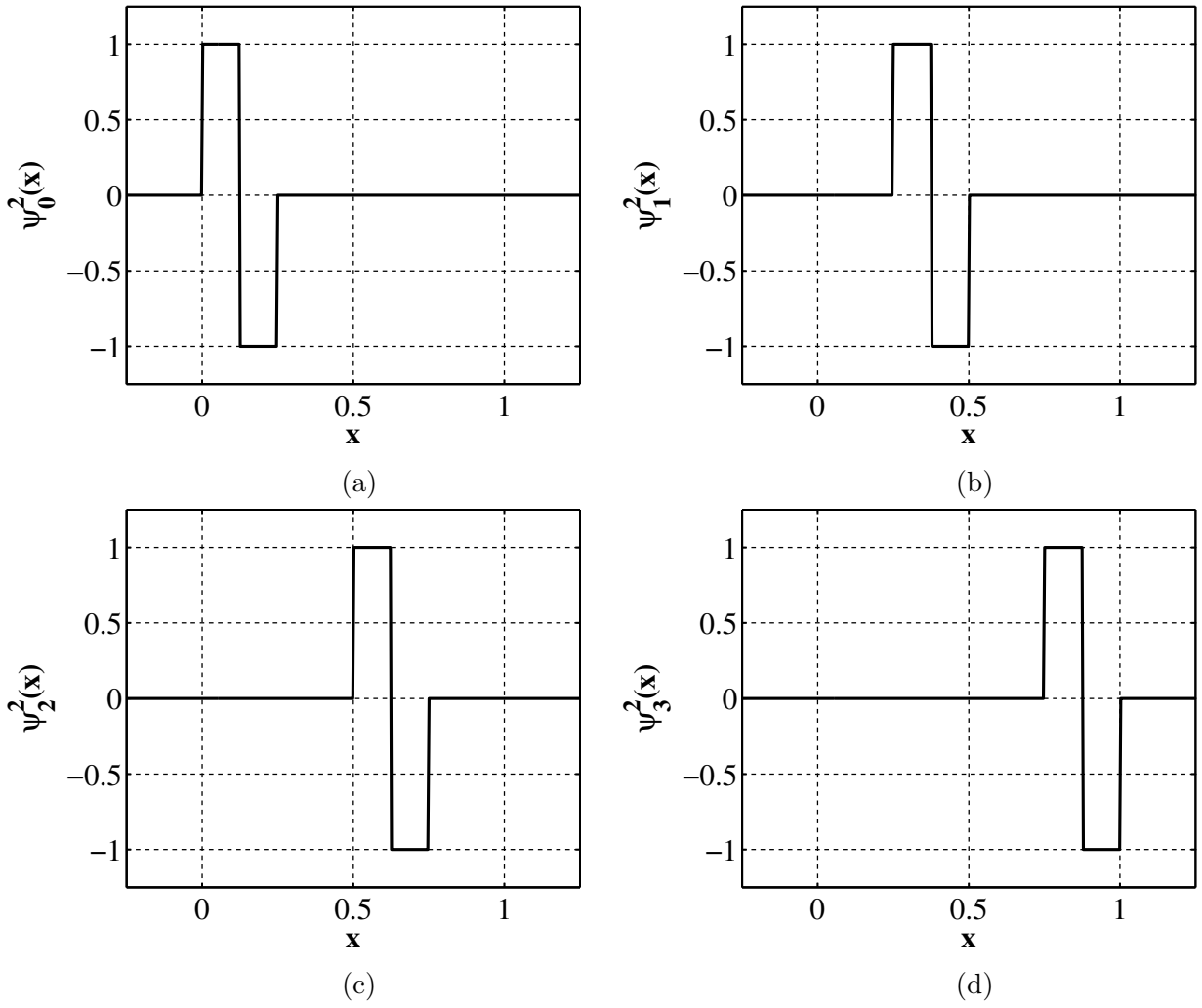


Figure 8.2: The wavelet functions that serve as the basis for W^2.

matrix of this size would require a large amount of memory and the time to perform the matrix-vector multiplication could be significant. In order to over come the performance downfalls of the matrix form of the transformation, a recursive algorithm described in [2] was used to generate the set of coefficients from the original signal without the matrix-vector multiplications. An overview of the algorithm implemented to convert the occupancy grid to the set of coefficients is described by Algorithm 8.3.

Algorithm 8.3 Recursive Haar wavelet decomposition from [2].

procedure HAARDECOMPOSITION($\mathbf{x}$)
    Data:
        $\mathbf{x}$ the input signal to be transformed, of size n

    Result:
        $\mathbf{y}$ the transformed version of $\mathbf{x}$

    $\mathbf{y} \leftarrow \mathbf{x}/\sqrt{n}$
    while $n > 1$ **do**
        HAARDECOMPOSITIONSTEP($\mathbf{y}(1:n)$)
        $n \leftarrow n/2$
    end while
end procedure

procedure HAARDECOMPOSITIONSTEP($\mathbf{x}$)
    Data:
        $\mathbf{x}$ the input signal to be transformed, of size n

    Result:
        $\mathbf{y}$ the transformed version of $\mathbf{x}$

    for $i \leftarrow 1, n/2$ **do**
        $\mathbf{y}(i) \leftarrow (\mathbf{x}(2i-1) + \mathbf{x}(2i))/\sqrt{2}$
        $\mathbf{y}(n/2+i) \leftarrow (\mathbf{x}(2i-1) - \mathbf{x}(2i))/\sqrt{2}$
    end for
end procedure

8.4. Experimental Results

To illustrate the performance of the proposed approach a set of experimental results generated using the FastSLAM AOG approach are presented. Instead of storing the raw occupancy grid, the algorithm represents the occupancy grid using an alternate basis and stores the occupancy grid using the set of coefficients that represent the grid in this basis. The alternate basis that was chosen to represent the occupancy grid is the Haar Wavelet basis.

The test vehicle used to run the algorithm is a small ground vehicle equipped with four Mechanum wheels ([83]). The vehicle was operated in small enclosed test environment 3m $\times$ 3.5m in size. The set of sensors attached to the test platform are a LiDAR ([64]), the custom designed laser based rangefinder previously described in Chapter 3, a downward facing camera that returns vehicle translations in the vehicles local frame using a correlation based template matching approach described in Chapter 4, a compass that provides the heading of the vehicle in the global frame, and a Stargazer indoor localization system ([66]). The Stargazer localization sensor is not by the FastSLAM AOG algorithm but rather it provides a baseline *true* path.

8.4.1. Results using LiDAR

The first set of experimental results were generated using the LiDAR attached to the test platform. A baseline vehicle path and occupancy grid were generated using the FastSLAM OG algorithm and are shown in Figure 8.3. The vehicle path and occupancy grid generated by the FastSLAM AOG algorithm are shown in Figure 8.4. To illustrate how the FastSLAM AOG estimate evolves over time a series of 9 evenly spaced path estimates are shown in Figure 8.5 and the corresponding occupancy grids are shown in Figure 8.6. In Figure 8.7 the

coefficients that correspond to the frames in Figure 8.6 are displayed sorted by magnitude. The sorting by magnitude was done in order to better illustrate how the full set of coefficients evolves during the experimental run. As seen from the coefficient snapshots early in the run when very little is known about the environment the magnitude of the coefficients are small. As the UMV explores the environment and new information about the surroundings is learned the magnitude of the coefficients grows. A comparison of the position error corresponding to the estimate generated by the FastSLAM OG algorithm and that produced by the FastSLAM AOG algorithm is shown in Figure 8.8. From this result it can be seen that while the two solutions do not match exactly, however the error for each of the algorithms is on the same order of magnitude and there is not significant difference between the two.

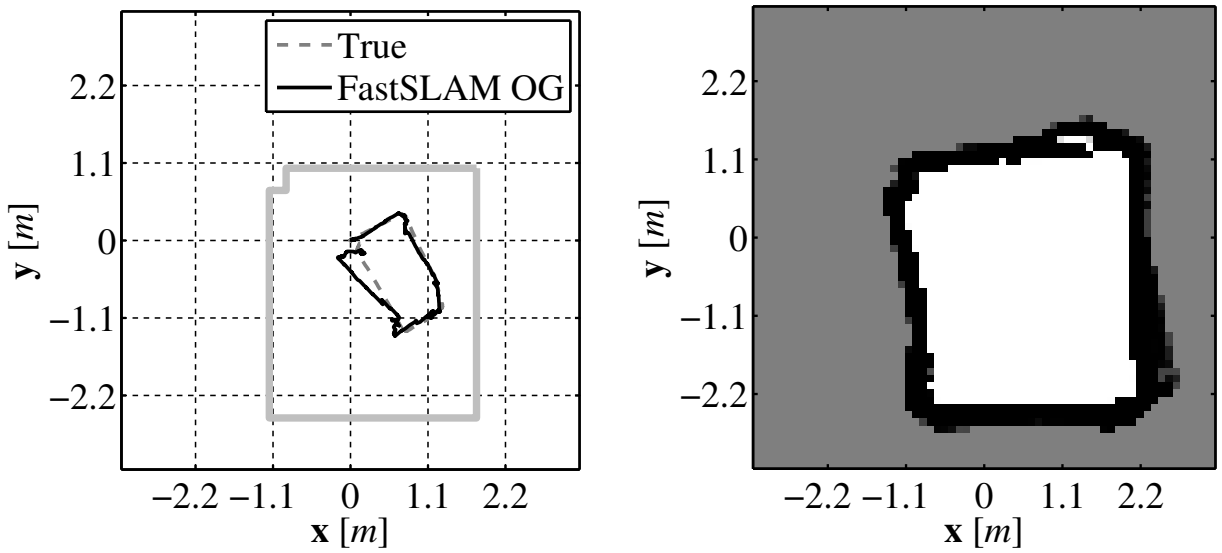
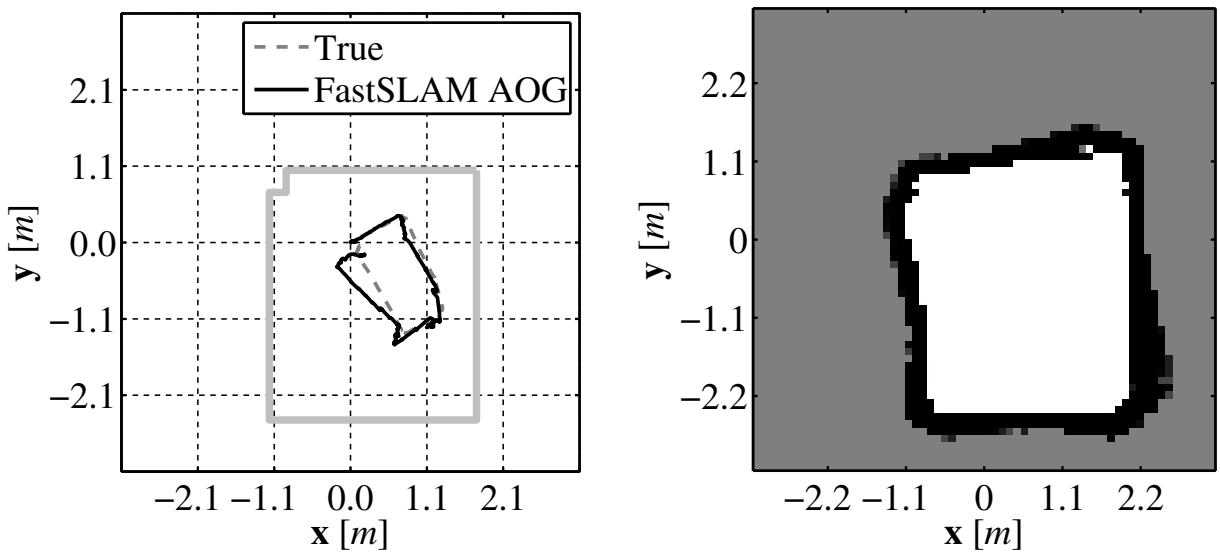


Figure 8.3: The FastSLAM OG produced path estimate (a) and map (b) that serve as a baseline generated with LiDAR.



(a) (b)

Figure 8.4: The FastSLAM AOG produced path estimate (a) and map (b) generated with LiDAR.

8.4.2. Results Using Laser Based Rangefinder

With the experimental results presented using the FastSLAM AOG algorithm along with the LiDAR attached to our test platform, we will now present the results of using the FastSLAM AOG algorithm with our custom designed laser based rangefinder from Chapter 3. As a baseline the UMV path estimate and map generated by the FastSLAM OG algorithm are shown in Figure 8.9. The final path and occupancy grid generated by the FastSLAM AOG algorithm using our custom rangefinder can be seen in Figure 8.10. As in the case of the LiDAR result a sequence of 9 images that illustrate how the estimate evolves over time are show for the path estimate Figure 8.11, the occupancy grid Figure 8.12, and the sorted coefficients that represented the occupancy grid in the Haar wavelet basis Figure 8.13. Finally, a comparison of the position error as a function of time for the FastSLAM OG and FastSLAM AOG algorithm is shown in Figure 8.14.

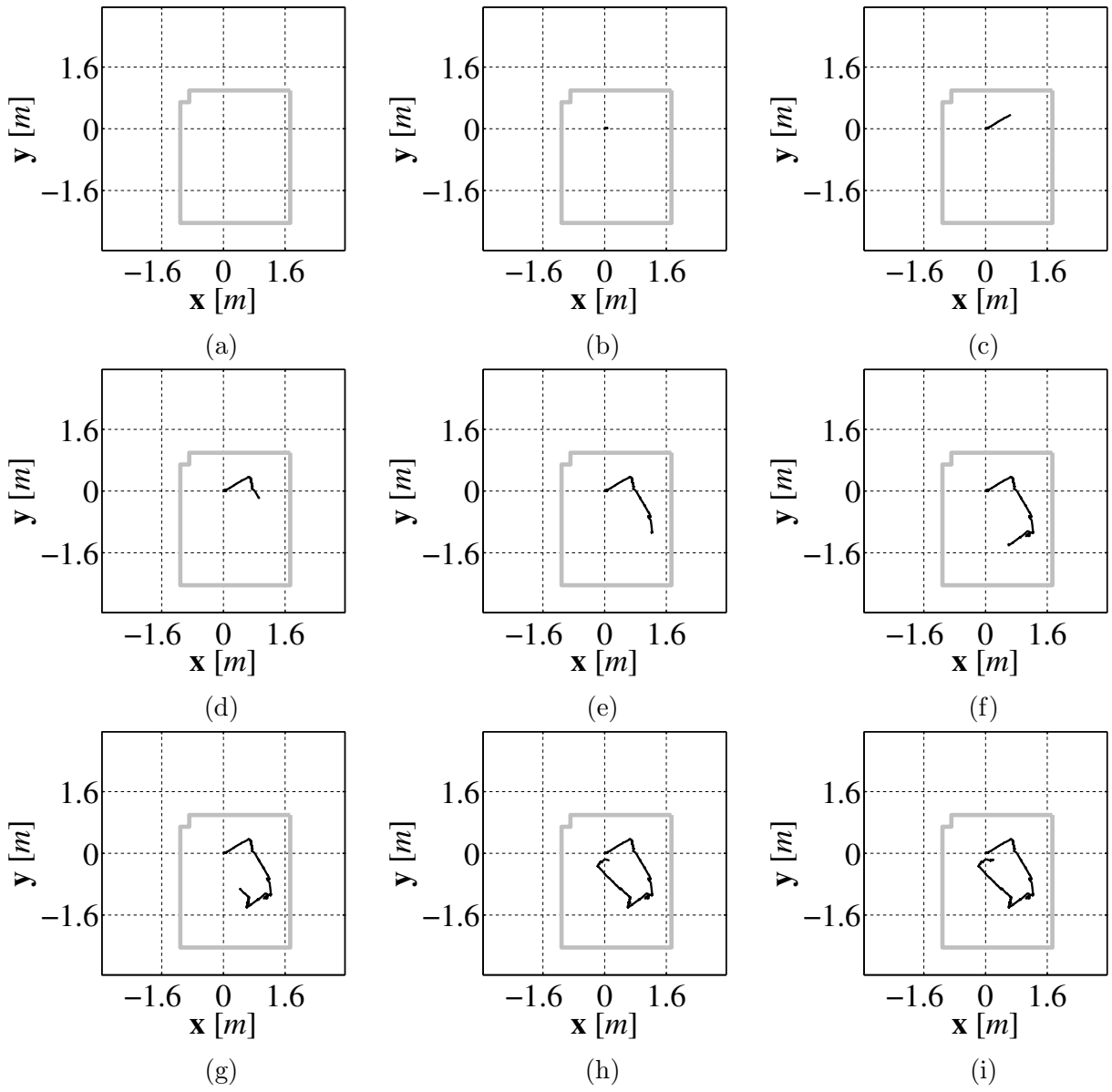


Figure 8.5: Evenly spaced sequence of path estimates produced by the FastSLAM AOG algorithm generated with LiDAR.

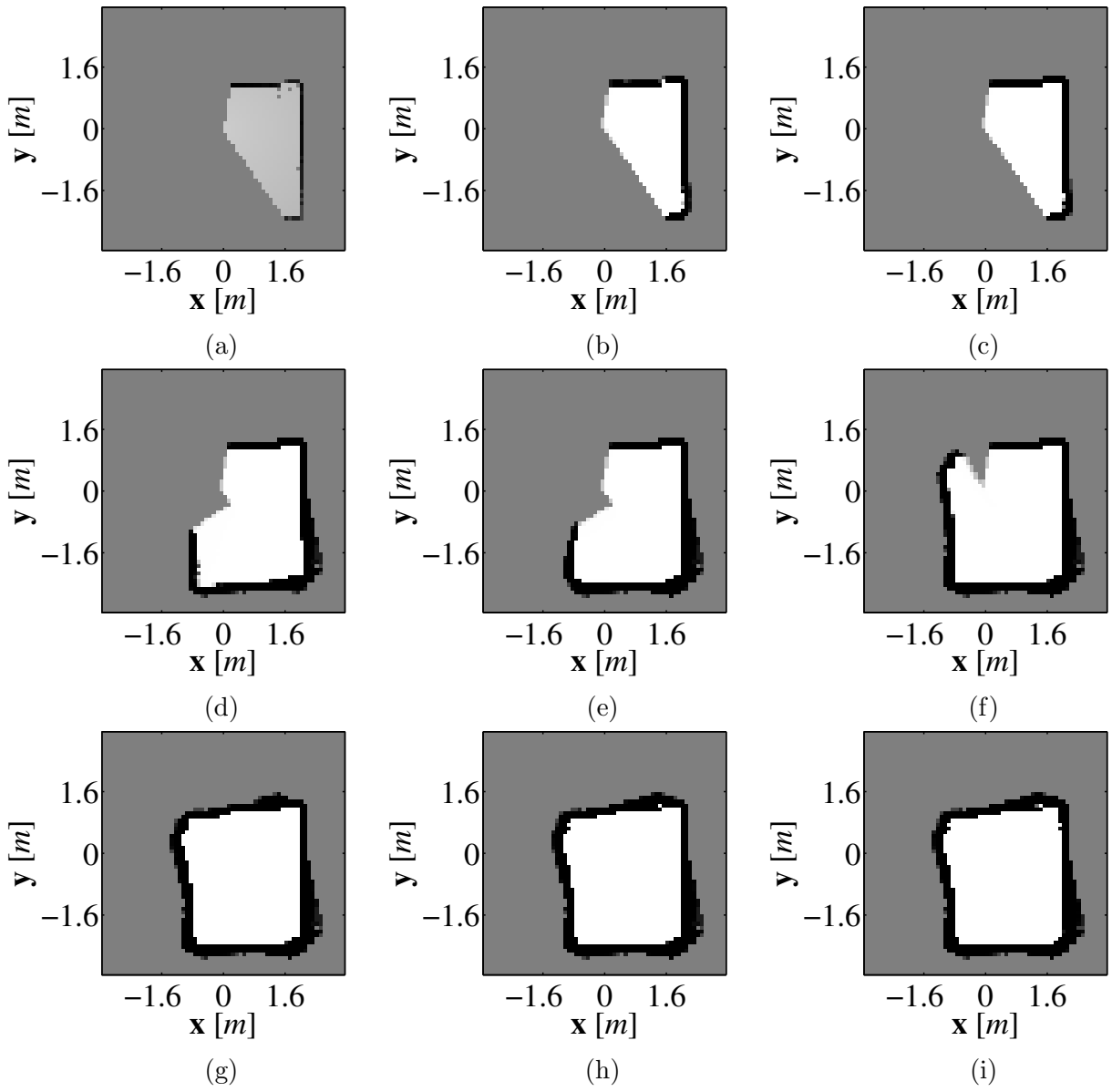


Figure 8.6: Evenly spaced sequence of map estimates produced by the FastSLAM AOG algorithm generated with LiDAR.

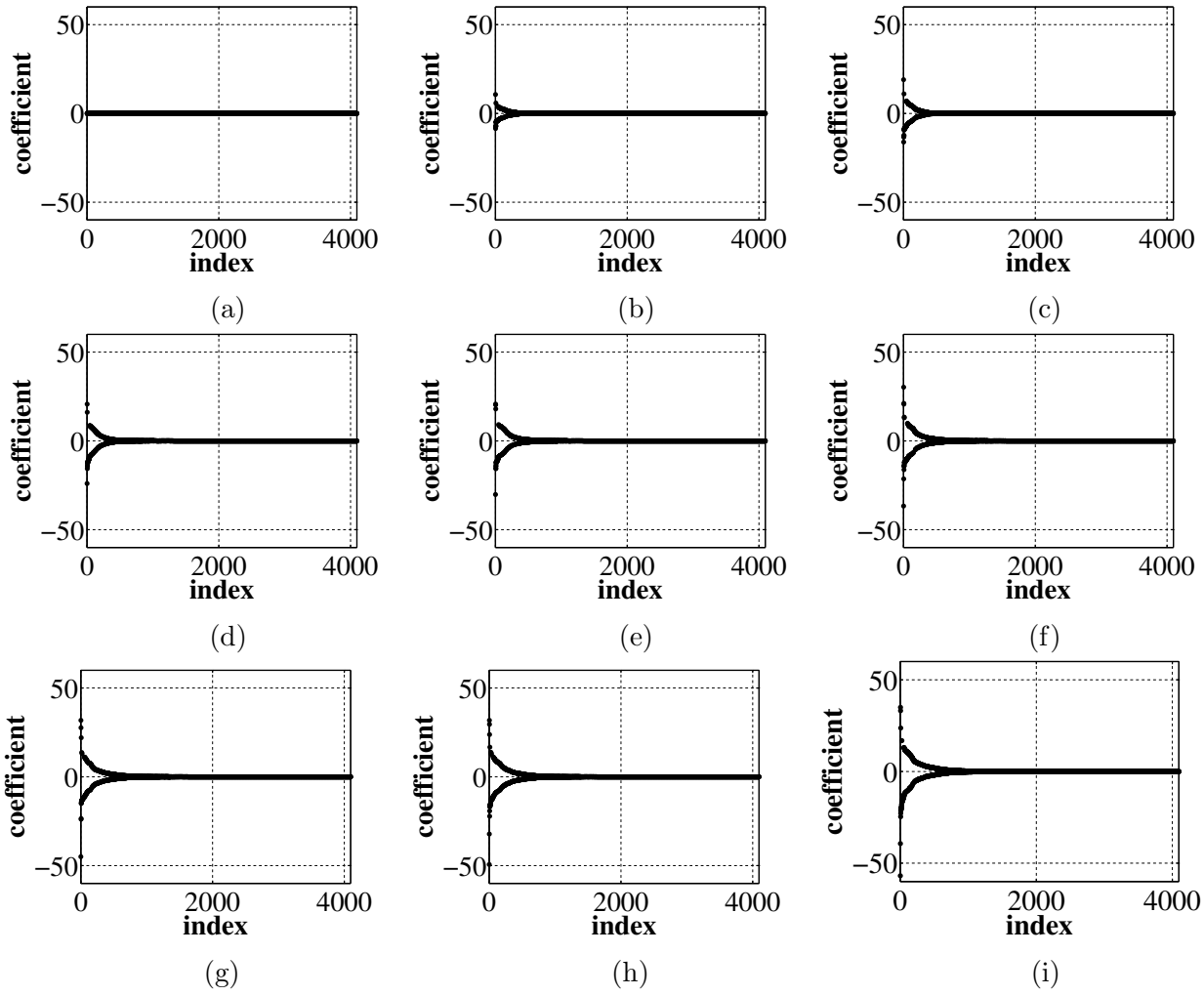


Figure 8.7: Evenly spaced sequence of occupancy grid coefficient sets produced by the FastSLAM AOG algorithm generated with LiDAR.

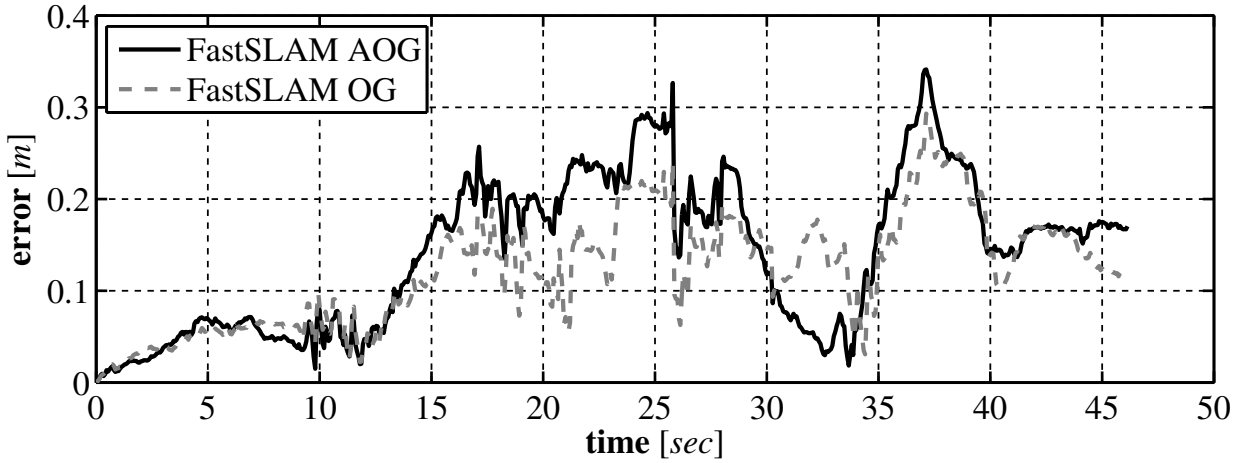


Figure 8.8: The error in the FastSLAM AOG position estimate over time generated with LiDAR.

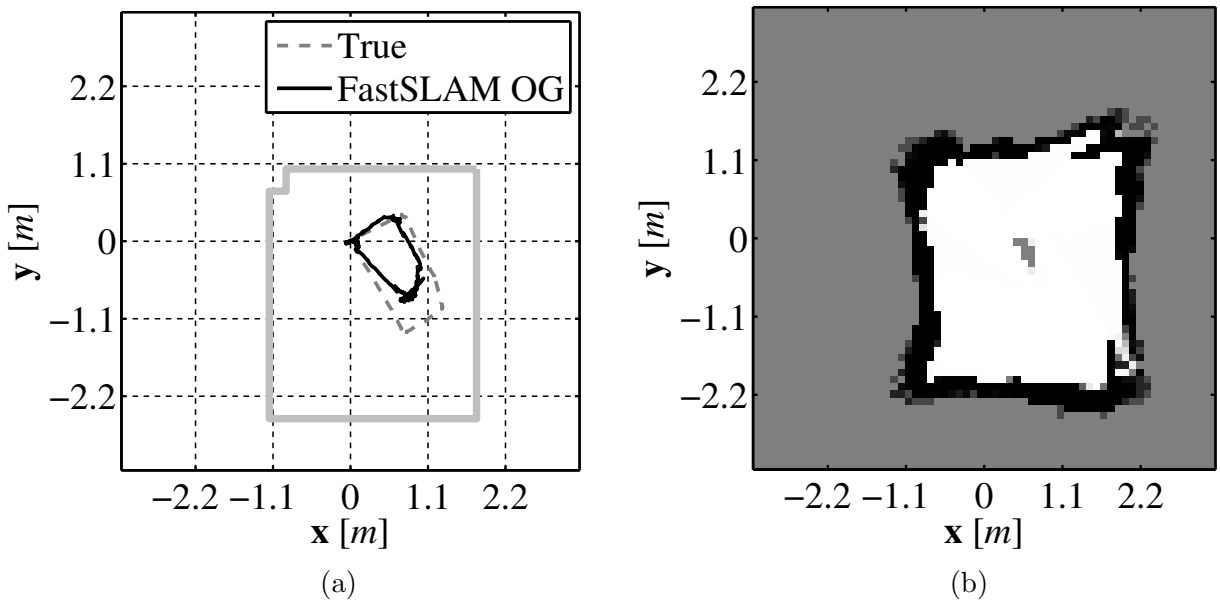


Figure 8.9: The FastSLAM OG produced path estimate (a) and map (b) that serves as a baseline generated with custom laser rangefinder.

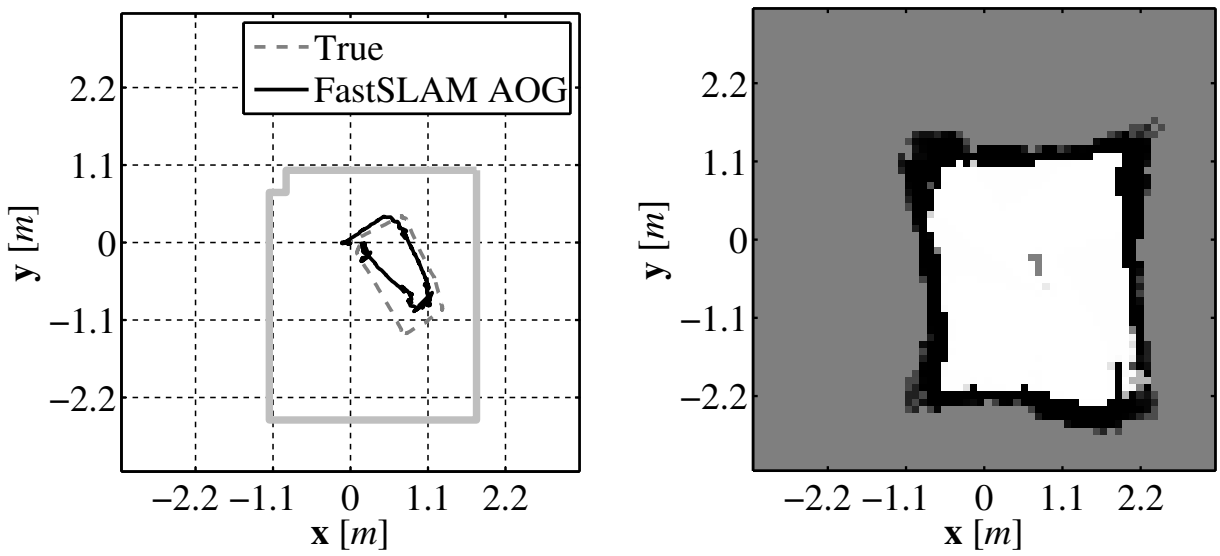


Figure 8.10: The FastSLAM AOG produced path estimate (a) and map (b) generated with custom laser rangefinder.

8.5. Conclusions

In this chapter the FastSLAM AOG algorithm was presented. The FastSLAM AOG algorithm is an extension of the FastSLAM OG algorithm presented in Chapter 7 that changes how the occupancy grid maintained by each particle is stored. In Section 8.2 the FastSLAM AOG algorithm was presented in detail. In Section 8.3 the Haar wavelet basis was presented and experimental results using the FastSLAM AOG algorithm along with the Haar wavelet basis were presented in Section 8.4 using a LiDAR sensor and our custom designed laser based rangefinder from Chapter 3. The FastSLAM AOG algorithm serves as the basis for the final SLAM algorithm that we will develop that implements the FastSLAM approach using compressed occupancy grids and that algorithm is developed in the following chapter.

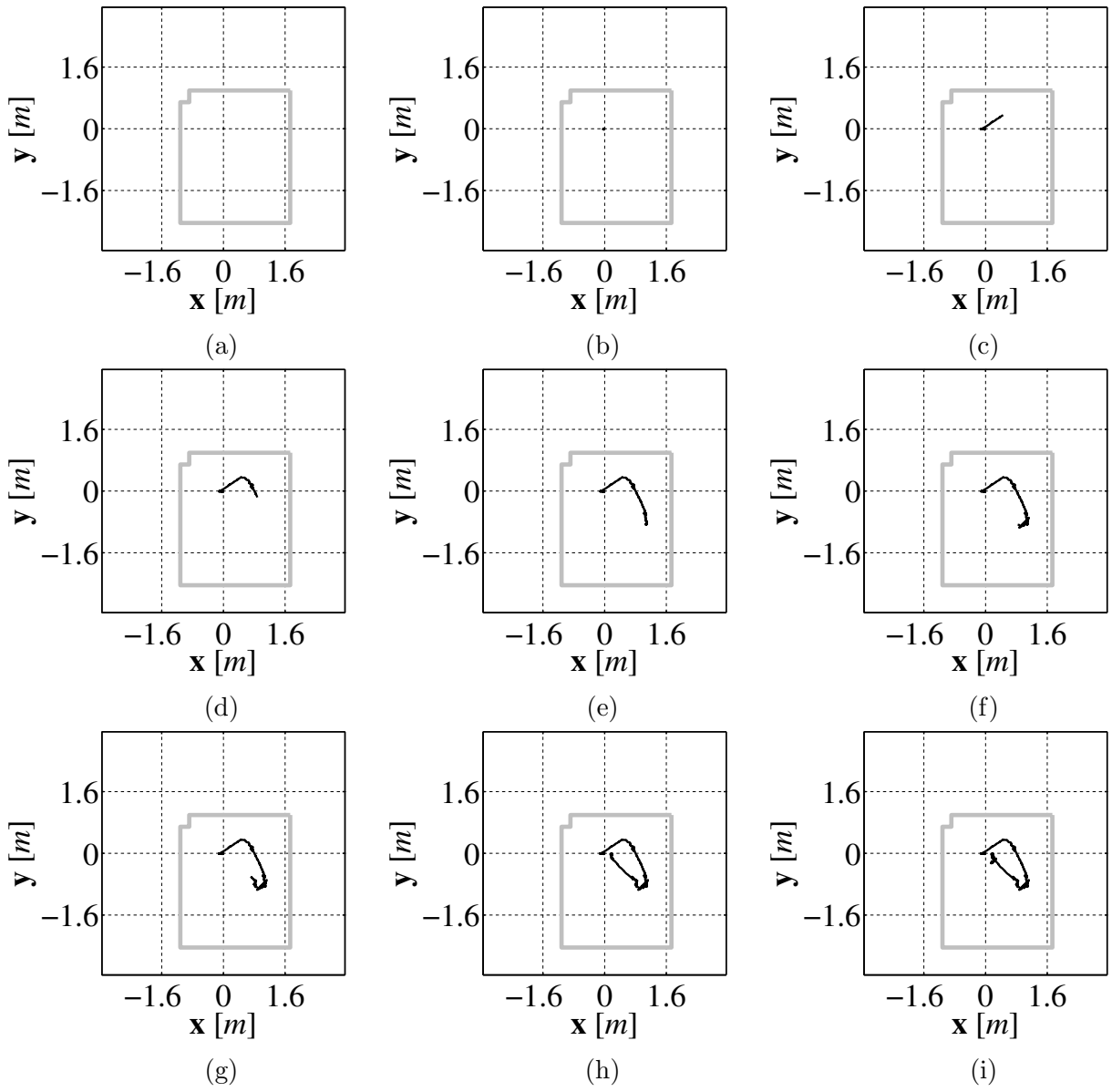


Figure 8.11: Evenly spaced sequence of path estimates produced by the FastSLAM AOG algorithm generated with custom laser rangefinder.

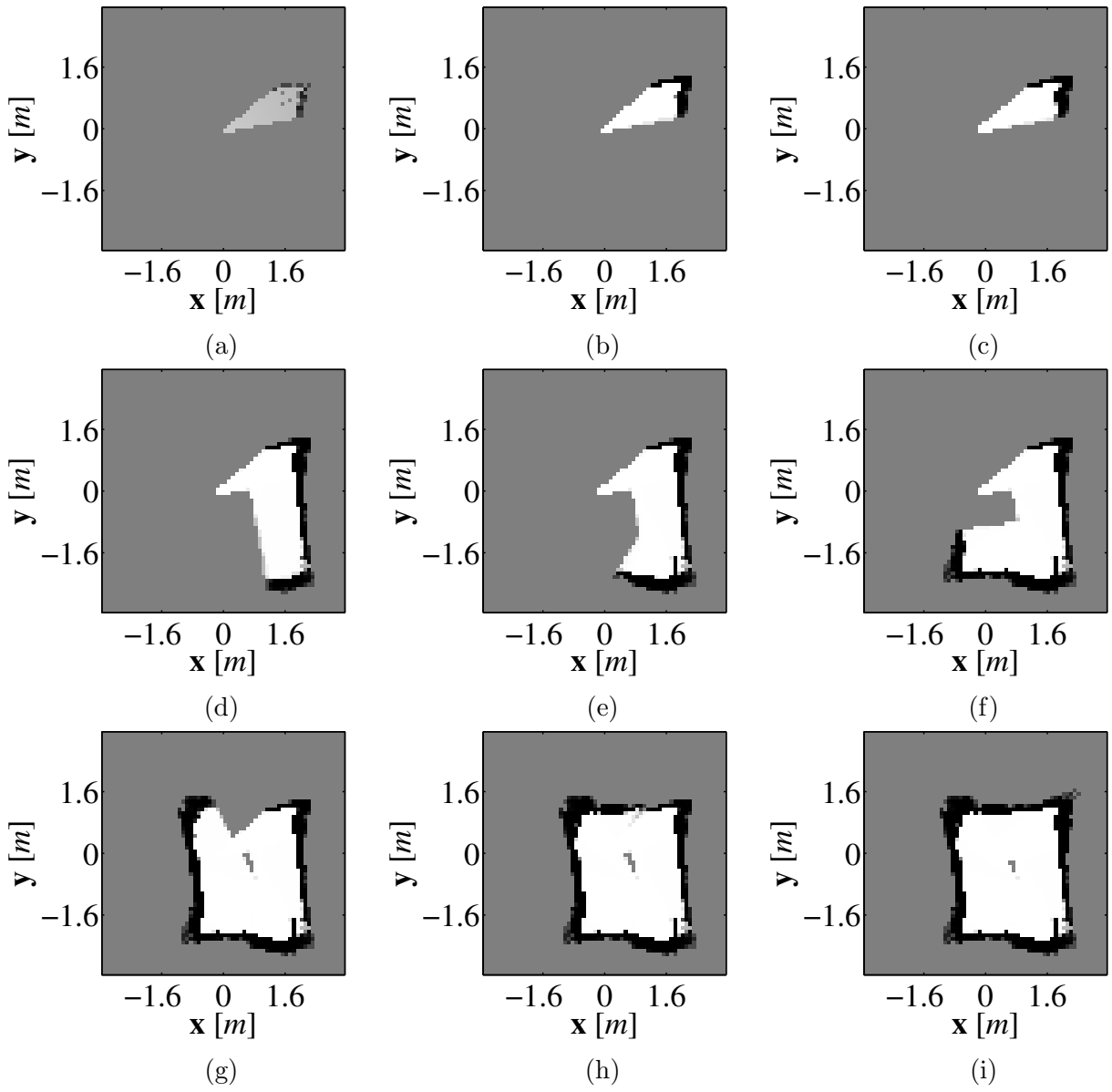


Figure 8.12: Evenly spaced sequence of map estimates produced by the FastSLAM AOG algorithm generated with custom laser rangefinder.

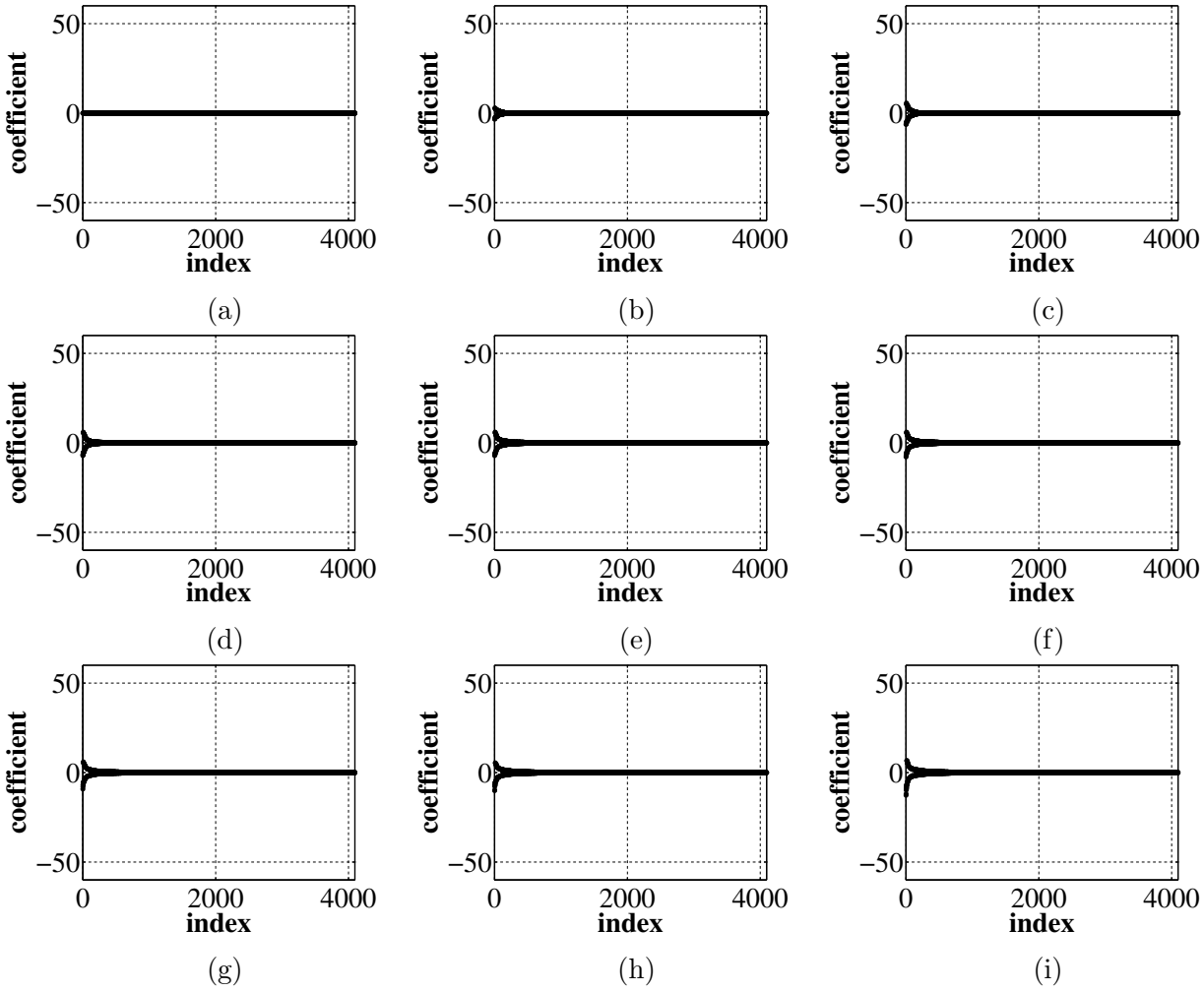


Figure 8.13: Evenly spaced sequence of occupancy grid coefficient sets produced by the FastSLAM AOG algorithm generated with custom laser rangefinder.

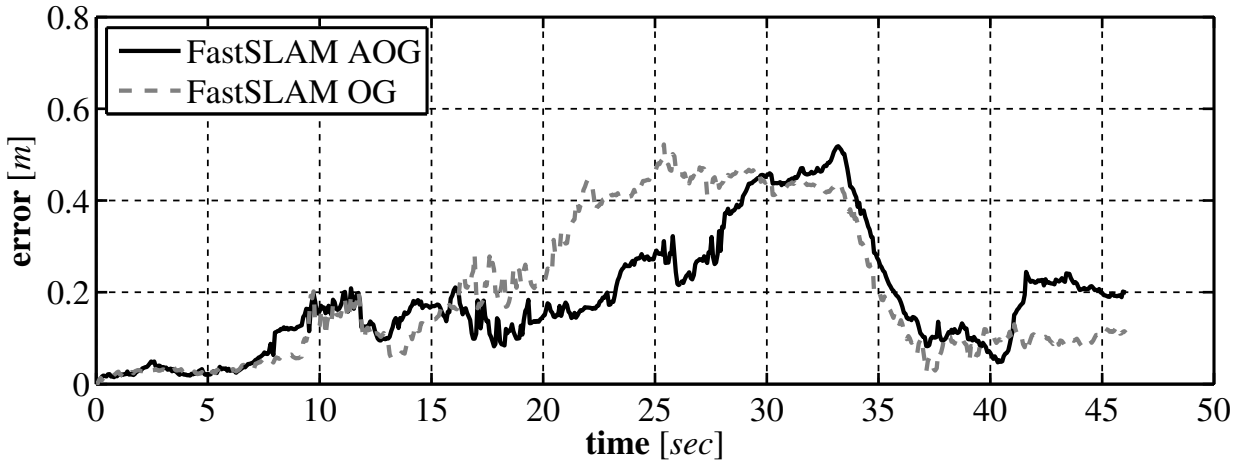


Figure 8.14: The error in the FastSLAM OG position estimate over time generated with custom laser rangefinder.

Chapter 9

FastSLAM With Compressed Occupancy Grids

9.1. Introduction

As discussed in the previous two chapters using occupancy grids in a particle filter framework to solve the SLAM problem can be memory intensive. In order to overcome this downfall in this chapter we develop an extension of the FastSLAM AOG algorithm introduced in the previous chapter that stores the occupancy grid in compressed form. In the literature there have been two primary approaches to reducing the amount of memory required when using particle filters and occupancy grids. In [84] the authors reduce the number of particles required to produce a good estimate by improving the distribution from which potential particles are sampled. The vehicle that they use is equipped with a LiDAR sensor which is much more accurate than the odometry sensors that are attached to their vehicle. Before new particles are generated the odometry data generated for their vehicle is corrected using the results of a scan matching procedure that is performed on two sequential LiDAR scans. By updating the proposal distribution from which particles are sampled, to make it more closely resemble the target distribution, they improve the performance by reducing the number of particle required. This is a good approach when the sensor used to observe

the environment is highly accurate, however, in our case our custom laser rangefinder can be significantly noisy, especially at large distance, so performing a proper scan matching procedure can be difficult. A second approach found in the literature is referred to as DP-SLAM ([37]). In this approach an alternate method of storing the occupancy grid for each particle is proposed. Instead of each particle maintaining its own copy of the occupancy grid there is a single occupancy grid for all particles. Each cell in the occupancy grid maintains a tree structure. Every time that a particle makes an observation that affects a cell in the occupancy grid a node is added to the tree structure containing the ID of the particle that made the observation and value of the observation. This approach is elegant and has been shown to be computationally efficient, however it adds additional complexity. One of the key advantages of the FastSLAM approach is its simplicity, thus the algorithm that we develop in this chapter decreases the amount of memory required to produce a good estimate my treating the problem as a data compression problem.

The remainder of this chapter is organized as follows. The FastSLAM algorithm that makes use of compressed occupancy grids (FastSLAM COG) is introduced in Section 9.2. In Section 9.3 four separate approaches for uncompressing the compressed occupancy grid are examined. Section 9.4 presents a set of performance improvements to some components of the algorithm. In Section 9.5 experimental results are provided and concluding remarks are presented in Section 9.6.

9.2. FastSLAM With Compressed Occupancy Grids

In general the goal of SLAM is to estimate the pose $\boldsymbol{\xi}_k$ of a UMV and the map, $\mathbf{M}$, of the unknown environment in which the UMV is operating at some time step $k > 0$. We present an extension of the FastSLAM algorithm that uses compressed occupancy grids to store the map. As opposed to many SLAM solutions that attempt to solve the online SLAM

problem, that is estimating the posterior that represents the pose of a UMV and map of the environment based on a set of control inputs and sensor measurements, the FastSLAM COG approach attempts to estimate the full SLAM posterior. The full SLAM posterior is given as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{9.1}$$

which is the distribution that represents the trajectory of a UMV $\boldsymbol{\xi}_{1:k}$ and the map of the environment $\mathbf{M}$ based on the set of control inputs $\mathbf{u}_{1:k}$ and observations $\mathbf{z}_{1:k}$. By estimating the full SLAM posterior (9.1) can be factored, using the property of conditional independence, into a pair of simpler terms which is given by

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{9.2}$$

where $p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)$ is the distribution that represents the UMV trajectory and the distribution that represents the map is denoted $p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)$. The FastSLAM COG approach estimates the factored SLAM posterior (9.2) using a Rao-Blackwellized particle filter. The Rao-Blackwellized particle filter is a version of a sampling importance resampling (SIR) ([76]) particle filter and it is the SIR particle filter that provides the format of the FastSLAM COG solution. The Rao-Blackwellized particle filter estimates the distribution representing a portion of the state, which in FastSLAM COG is the UMV trajectory $\boldsymbol{\xi}_{1:k}$, using a particle filter, and each particle in the particle filter maintains its own estimate of the remainder of the state, which is the map $\mathbf{M}$. In the FastSLAM COG algorithm the occupancy grid is stored using a compressed form of the occupancy grid and the details of this representation will be presented in the following section. As with the FastSLAM OG and FastSLAM AOG algorithms, the FastSLAM COG algorithm is performed in several steps and an overview of the algorithm is presented in Algorithm 9.1.

Algorithm 9.1 Overview of the FastSLAM COG algorithm

procedure FastSLAMCOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
    Data:
        $\mathcal{X}_{k-1}$ the particle set from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of measurements

    Result:
        $\mathcal{X}_k$ the particle set from the current time step

    $\overline{\mathcal{X}}_k \leftarrow \{\}$
    for $p \leftarrow 1, n$ **do**
        $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \;\middle|\; \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
        $\hat{\mathbf{c}}_k^{[p]} = \text{UNCOMPRESS}(\mathbf{y}_{k-1}^{[p]})$
        $w^{[p]} \leftarrow \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \boldsymbol{\Phi}\hat{\mathbf{c}}_k^{[p]}\right)$
        $m = \text{LENGTH}(\hat{\mathbf{c}}_{k-1}^{[p]})$
        $\alpha_k^{[p]} = \text{AlphaConstruction}(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k, m)$
        $\mathbf{y}_k^{[p]} = \mathbf{H}\left[\hat{\mathbf{c}}_k^{[p]} + \boldsymbol{\Phi}^{\mathrm{T}}\left[\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta}\right]\right]$
        $\overline{\mathcal{X}}_k^{[p]} \leftarrow \left[\begin{array}{ccc} \boldsymbol{\xi}_k^{[p]} & \mathbf{y}_k^{[p]} & w^{[p]} \end{array}\right]$
    end for
    $\mathcal{X}_k \leftarrow \{\}$
    for $j = 1, n$ **do**
        sample with replacement from select i with probability proportional to $w^{[p]}$
        $\mathcal{X}_k^{[p]} \leftarrow \overline{\mathcal{X}}_k^{[p]}$
    end for
end procedure

9.2.1. Compressed Occupancy Grid Representation

Before discussing the updated SLAM algorithm, we will first address the compressed form of the occupancy grid. To compress $\mathbf{M}_k^{[p]}$ we first reshape the occupancy grid so that the $h \times w$ matrix becomes a column vector containing $n = wh$ elements and we refer to the vector form of the occupancy grid as $\mathbf{m}_k^{[p]} \triangleq \text{vec}(\mathbf{M}_k^{[p]})$. The reshaping process is performed in such a way that the cell in the original occupancy grid located at (i, j) can be accessed in the vector form at l where $l = iw + j$. In order to compress the occupancy grid without major degradation, we assume that the occupancy grid can be represented in an alternate basis in which it is sparse. The relationship between the two different representation is given as

$$\mathbf{m}_k^{[p]} = \mathbf{\Phi}\mathbf{c}_k^{[p]}, \tag{9.3}$$

where $\mathbf{\Phi}$ is a transformation matrix that defines the relationship and $\mathbf{c}_k^{[p]}$ is the set of coefficients that represent the occupancy grid in the alternate basis. In order for the occupancy grid to be compressible, $\mathbf{c}_k^{[p]}$ must be sparse ([85]), that is $\#\{\text{supp}\left(\mathbf{c}_k^{[p]}\right)\} < n$, where $\text{supp}(\cdot)$ is the support operator and $\#$ denotes the cardinality of the set. This means that in order for us to be able to compress the occupancy grid we must find a basis in which the coefficient vector is composed of a large number of zeros. If this condition is met then the occupancy grid for a given particle can be compressed using

$$\mathbf{y}_k^{[p]} = \mathbf{H}\mathbf{c}_k^{[p]}, \tag{9.4}$$

where $\mathbf{H}$ is a matrix that performs the compression by selecting a subset of coefficient data that is smaller than the amount of data in the complete signal.

The FastSLAM COG algorithm is a modification of the FastSLAM AOG algorithm presented in Chapter 8. Since we are now maintaining the compressed form of the occupancy

grid, the pth particle in the particle set is defined as

$$\mathcal{X}_k^{[p]} \triangleq \begin{bmatrix} \boldsymbol{\xi}_k^{[p]} & \mathbf{y}_k^{[p]} \end{bmatrix}. \tag{9.5}$$

9.2.2. Pose Sampling

As in all of the previously discussed FastSLAM algorithms in the first step of the FastSLAM COG algorithm (Line 12 in Algorithm 9.1) a new particle pose is generated and added to a set of temporary particles, $\overline{\mathcal{X}}_k$. The new pose for the pth particle is generated by sampling from the probabilistic motion model of the UMV

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right). \tag{9.6}$$

The probabilistic motion model of the UMV generates a new pose based on the state transition model of the UMV, the pose of the pth particle at time step $k-1$, the current control input $\mathbf{u}_k$, and the characteristics of the noise on $\mathbf{u}_k$. The same sampling procedure as used in the FastSLAM, FastSLAM OG, and FastSLAM AOG algorithms is used and the details are given in Section B.2. The result is that the particles in $\overline{\mathcal{X}}_k$ generated in the sampling process are distributed according to

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right), \tag{9.7}$$

which is referred to as the *proposal distribution*. The actual distribution which we are attempting to estimate is referred to as the *target distribution* and given as

$$p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right). \tag{9.8}$$

The proposal distribution is transformed to the target distribution in the last stage of the algorithm, the resampling stage (Lines 21-24 in Algorithm 9.1). Before the resampling process can occur, $\mathbf{z}_k$ must be incorporated into the estimate, which is achieved in the calculation of the particle's importance weight.

9.2.3. Importance Weight Calculation

In the second step of the algorithm an importance weight for each particle is generated. The need for an importance weight for each particle comes from our use of a SIR particle filter and as seen in the previous section we are sampling particles from a distribution that does not match the distribution that we are attempting to estimate. From [77] if we are unable to directly sample from the distribution that we are attempting to estimate, then if the importance weight for each particle is given as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \tag{9.9}$$

and particles are drawn from $\overline{\mathcal{X}}_k$ with replacement and added to $\mathcal{X}_k$ with a probability proportional to $w_k^{[p]}$ then $\mathcal{X}_k$ will approximate the target distribution and the quality of the approximation will improve as the number of particles increases.

Using (9.9) along with the previously discussed distributions, (9.7) and (9.8), the importance weight of each particle simplifies to ([81])

$$w_k^{[p]} = \frac{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)}{p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right)} = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right), \tag{9.10}$$

where $p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$ is the probabilistic measurement model of the sensor being used by the UMV and $\eta > 0$ is a normalizing factor. The probabilistic measurement model is a measure of how well the current set of sensor measurements matches up with what we expect based on the current pose and map of the particle. In calculating the probabilistic measurement model, the algorithm must have access to the raw occupancy grid values so that an expected sensor measurement can be generated. Based on this requirement, the compressed occupancy grid coefficients are first uncompressed. For now a general uncompression operator denoted $\textbf{uncompress}\left(\cdot\right)$ is used, however the details of several uncompression methods are explored in detail in Section 9.3. Once the compressed coefficients have been uncompressed, the occupancy grid can be extracted using (9.3). Thus, the importance weight

calculation for the pth particle is given as

$$w_k^{[p]} = \eta p \left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \Phi^{\mathrm{T}} \mathbf{uncompress} \left(\mathbf{y}_{k-1}^{[p]} \right) \right), \tag{9.11}$$

The importance weight calculation can be found in Algorithm 9.1 on Line 14 .

9.2.4. Compressed Occupancy Grid Update

As previously seen in Chapter 7 Section 7.2 the standard approach to updating the occupancy grid is to examine every cell in the occupancy grid and if a given cell falls within the perceptual range of the sensor attached to the UMV then the corresponding cell of the occupancy grid is updated according to

$$\mathbf{M}_k^{[p]}(i,j) = \mathbf{M}_{k-1}^{[p]}(i,j) + \log \frac{p\left(\mathbf{M}(i,j) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(l) \right)}{1 - p\left(\mathbf{M}(i,j) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(l) \right)} - l_0. \tag{9.12}$$

Using (9.12) as our basis, we rewrite the occupancy grid update in vector form. If we assume that the full occupancy grid is stored in vector form, then the occupancy grid update can be performed using

$$\mathbf{m}_k^{[p]} = \mathbf{m}_{k-1}^{[p]} + \boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta}, \tag{9.13}$$

where $\boldsymbol{\alpha}_k^{[p]}$ is constructed using Algorithm 9.2 and $\boldsymbol{\beta}$ is vector with every element containing l_0.

Algorithm 9.2 first initializes $\boldsymbol{\alpha}_k^{[p]}$ using l_0. This is so that the value associated with cells that do not fall into the perceptual range of the sensor remains unchanged. For cells that fall into the perceptual range of the sensor, their $\boldsymbol{\alpha}$ value is set as the log odds form of the inverse sensor model and it is this value that either increases or decreases the likelihood that the cell is occupied. Equation (9.13) updates the occupancy grid in vector form, however we need to modify it so that it updates the occupancy grid which is stored using an alternative basis. Incorporating (9.3), the vector form of the occupancy grid update in the alternate

Algorithm 9.2 Construction of the occupancy grid update vector $\boldsymbol{\alpha}_k^{[p]}$.

procedure AlphaConstruction($\boldsymbol{\xi}_k^{[p]}$, $\mathbf{z}_k$, n)
    Data:
        $\boldsymbol{\xi}_k^{[p]}$ the pose estimate of the current particle
        $\mathbf{z}_k$ the current set of measurements, of length m
        n the size of the vector form of the occupancy grid

    Result:
        $\boldsymbol{\alpha}_k^{[p]}$ the vector containing the occupancy grid update components

    for $i = 1, n$ **do**
        $\boldsymbol{\alpha}(i) = l_0$
    end for
    for $i = 1, n$ **do**
        for $j = 1, m$ **do**
            if i corresponds to a cell that lies in the perceptual range of $\mathbf{z}(j)$ **then**
                $\boldsymbol{\alpha}(i) = \log \frac{p\left(\mathbf{m}(i) \,\middle|\, \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(j)\right)}{1 - p\left(\mathbf{m}(i) \,\middle|\, \boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(j)\right)}$
            end if
        end for
    end for
end procedure

basis becomes

$$\mathbf{c}_k^{[p]} = \mathbf{c}_{k-1}^{[p]} + \mathbf{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}_k^{[p]} - \mathbf{\Phi}^{\mathrm{T}} \boldsymbol{\beta}, \tag{9.14}$$

where $\mathbf{\Phi}^{\mathrm{T}}$ is used to convert $\boldsymbol{\alpha}_k^{[p]}$ and $\boldsymbol{\beta}$ to the alternate basis. In (9.14) $\mathbf{\Phi}^{\mathrm{T}}$ is used to change basis as opposed to Φ^{-1} because we assume that our choice of basis is orthonormal so $\mathbf{\Phi}^{-1} = \mathbf{\Phi}^{\mathrm{T}}$, however if the alternative basis that is selected is not orthonormal then $\mathbf{\Phi}^{-1}$ should replace $\mathbf{\Phi}^{\mathrm{T}}$ in (9.14). We can now simplify (9.14) by grouping the two components that are multiplied by $\mathbf{\Phi}^{\mathrm{T}}$ to reduce the number of matrix-vector multiplications, this yields

$$\mathbf{c}_k^{[p]} = \mathbf{c}_{k-1}^{[p]} + \mathbf{\Phi}^{\mathrm{T}} \left[\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta} \right]. \tag{9.15}$$

Equation (9.15) updates the occupancy grid in the alternate basis so, the final step is to incorporate compression into the update. The occupancy grid maintained by each particle contains the compressed set of basis of coefficients, thus (9.15) can be updated using our generic uncompression operator to yield

$$\mathbf{c}_k^{[p]} = \mathbf{uncompress}\left(\mathbf{y}_{k-1}^{[p]} \right) + \mathbf{\Phi}^{\mathrm{T}} \left[\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta} \right]. \tag{9.16}$$

Finally, the basis coefficients used to represent the occupancy grid must be re-compressed in order to be stored by the particle. Including the final compression the compressed occupancy grid update can be performed according to

$$\mathbf{y}_k^{[p]} = \mathbf{H} \left[\mathbf{uncompress}\left(\mathbf{y}_{k-1}^{[p]} \right) + \mathbf{\Phi}^{\mathrm{T}} \left[\boldsymbol{\alpha}_k^{[p]} - \boldsymbol{\beta} \right] \right], \tag{9.17}$$

which is seen in Algorithm 9.1 Line 17 .

9.2.5. Resampling

After the occupancy grid for each particle in $\overline{\mathcal{X}}_k$ has been updated, the final step of the algorithm is the resampling process. During the resampling process particles are drawn

with replacement from $\overline{\mathcal{X}}_k$ with a probability proportional to $w^{[p]}$. This resampling step takes the set of temporary particles that are distributed according to (9.7) and ensures that there is a high probability that $\mathcal{X}_k$ is distributed according to (9.8). The details of the approach that we use to perform the resampling are provided in Appendix B Section B.5. In the section the FastSLAM COG algorithm was developed using a generic uncompression operator, **uncompress** $(\cdot)$, in the following section four specific uncompression approach are presented in detail.

9.3. Compressed Signal Reconstruction Methods

As previously described we would like to store a compressed version of the occupancy grid as opposed to the complete occupancy grid to decrease the memory requirements of the algorithm. In order to make use of the compressed occupancy grid, a method for reconstructing the full occupancy grid from the compressed version must be selected. In this section several reconstruction methods are examined with the goal of selecting an approach to be used in FastSLAM COG. In FastSLAM COG we are attempting to compress the vector form of an occupancy grid, however in this section we will examine methods for reconstructing a generic discrete signal, $\mathbf{x} \in \mathbb{R}^n$ that has been compressed.

We represent $\mathbf{x}$ using some alternate basis in which it is sparse and the relationship between $\mathbf{x}$ and its sparse representation is given by

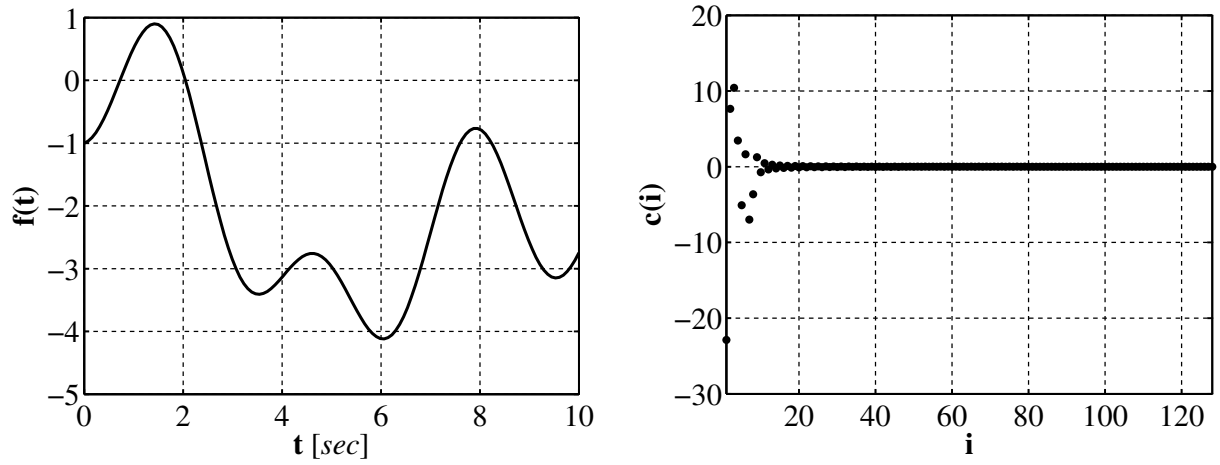
$$\mathbf{x} = \mathbf{\Phi c}. \tag{9.18}$$

where $\mathbf{c} \in \mathbb{R}^n$ is a vector of coefficients that represent $\mathbf{x}$ in the alternate basis and $\mathbf{\Phi} \in \mathbb{R}^{n \times n}$ is a matrix of basis functions that determines the relationship between $\mathbf{x}$ and $\mathbf{c}$. The discrete signal used in this section for testing is displayed in Figure 9.1a and the set of coefficients used to represent the signal in the discrete cosine basis ([86]) can be seen in Figure 9.1b.

The test signal is generated by evaluating

$$f(t) = \sin(t) - \cos(2t) - 3\sin(0.25t), \tag{9.19}$$

at 128 evenly spaced points in $[0, 10]$.



(a) (b)

Figure 9.1: (a) The test function given by (9.19) and (b) the test function represented in the discrete cosine basis.

If $\mathbf{c}$ is sparse then $\mathbf{x}$ can be compressed, this sparseness can be seen by examining Figure 9.1b and noticing that a large number of the coefficients are zero. The compression is performed by selecting a subset of the coefficient information using

$$\mathbf{y} = \mathbf{H}\mathbf{c}, \tag{9.20}$$

where $\mathbf{H} \in \mathbb{R}^{m \times n}$ is the compression matrix and $m \in \mathbb{N}$ is the size of the compressed signal with $m < n$.

9.3.1. l_2 Reconstruction (l_2)

One of the most simple methods for reconstructing a compressed signal is to reconstruct the signal while minimizing the error between the original and reconstructed signals as

measured by the l_2 norm, where the l_2 norm of a signal $\mathbf{x} \in \mathbb{R}^n$ is defined as

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} \mathbf{x}(i)^2}. \tag{9.21}$$

In examining (9.20) if the size of the compressed signal is the same as the original then the reconstruction can be performed according to

$$\mathbf{c} = \mathbf{H}^{-1}\mathbf{y}. \tag{9.22}$$

In reality the system is underdetermined due to $m < n$. A common solution to problems of this form use the Moore-Penrose pseudoinverse of $\mathbf{H}$.

The Moore-Penrose pseudoinverse ([87, 88]) is used to find the solution that minimizes the l_2 error for systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{9.23}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{A} \in \mathbb{R}^{m \times n}$. The solution to the system found using the pseudoinverse is defined as

$$\mathbf{z} = \mathbf{A}^{\dagger}\mathbf{b}, \tag{9.24}$$

where $\mathbf{A}^{\dagger}$ is the pseudoinverse of $\mathbf{A}$ and it minimizes the l_2 error so that

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \geq \|\mathbf{A}\mathbf{z} - \mathbf{b}\|_2, \tag{9.25}$$

for all $\mathbf{x} \in \mathbb{R}^n$ if $\mathbf{x}$ is defined as

$$\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{b} + \left(\mathbf{I} - \mathbf{A}^{\dagger}\mathbf{A}\right)\mathbf{w}, \tag{9.26}$$

for any vector $\mathbf{w} \in \mathbb{R}^n$.

Using the pseudoinverse of $\mathbf{H}$ a set of reconstructed coefficients is generated according to

$$\hat{\mathbf{c}} = \mathbf{H}^{\dagger}\mathbf{y}, \tag{9.27}$$

and the reconstructed value of the signal is generated from

$$\hat{\mathbf{x}} = \mathbf{\Phi}\hat{\mathbf{c}}. \tag{9.28}$$

The results of the l_2 reconstruction using four separate amounts of signal information: 25%, 50%, 75%, and 95% are presented in Figure 9.2. The form selected for $\mathbf{H}$ needed to obtain such level of compression is an orthonormal gaussian random matrix as it possesses the RIP property whose importance will be discussed in the following section.

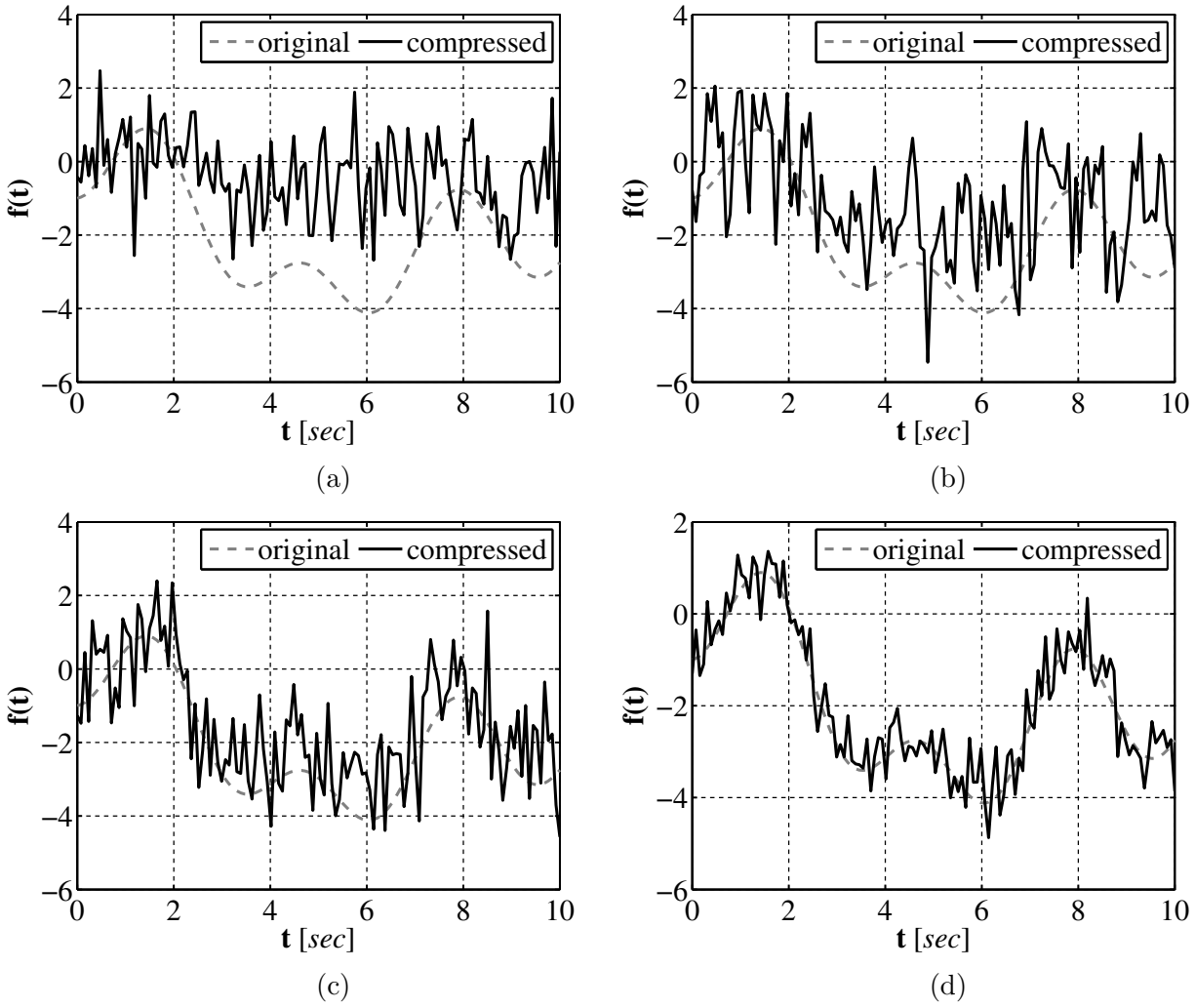


Figure 9.2: The test function reconstructed by minimizing the l_2 error for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data.

9.3.2. Compressed Sensing Reconstruction (CS)

The second reconstruction algorithm selected is based on an approach referred to in the literature as Compressed Sensing ([48]). Compressed sensing allows for sparse signals to be reconstructed using fewer measurements that would be required according to the Nyquist/Shannon sampling principle. In compressed sensing the full set of sparse coefficients can be reconstructed from the compressed set by solving

$$\min \|\hat{\mathbf{c}}\|_0 \quad \text{subject to} \quad \|\mathbf{y} - \mathbf{H}\hat{\mathbf{c}}\|_2^2 \leq \epsilon, \tag{9.29}$$

for some sufficiently small ϵ. In (9.29) $\|\hat{\mathbf{c}}\|_0 \triangleq \#\{\text{supp}(\hat{\mathbf{c}})\}$ and is a count of the number of non-zero elements in the vector $\hat{\mathbf{c}}$. Solving (9.29) is NP-hard and the solution cannot be found efficiently. A key finding of compressed sensing is that if the measurement matrix, $\mathbf{H}$, is constructed in such a way that it obeys the Restricted Isometry Property (RIP), which is defined as

$$(1-\delta)\|\mathbf{c}\|_2^2 \leq \|\mathbf{H}\mathbf{c}\|_2^2 \leq (1+\delta)\|\mathbf{c}\|_2^2, \tag{9.30}$$

for some small $\delta \geq 0$, then $\|\cdot\|_0$ in (9.29) can be replaced by $\|\cdot\|_1$ resulting in

$$\min \|\hat{\mathbf{c}}\|_1 \quad \text{subject to} \quad \|\mathbf{y} - \mathbf{H}\hat{\mathbf{c}}\|_2^2 \leq \epsilon. \tag{9.31}$$

As opposed to being NP-hard to solve, (9.31) is a constrained convex optimization problem and can be solved efficiently using one of several available software packages with Lasso ([89]), NESTA ([90]), and l1 magic ([91]) being some examples. As in the l_2 case once the complete set of coefficients have been estimated using the compressed set, an estimate of the original signal is generated according to (9.28).

In order to examine the performance of the Compressed Sensing reconstruction approach, the test signal is reconstructed using the same four amounts of data: 25%, 50%, 75%, and 95% with the same orthonormal gaussian random matrix being used to perform the

compression. Equation (9.31) is solved using the NESTA software library and the results of the reconstruction are shown in Figure 9.3.

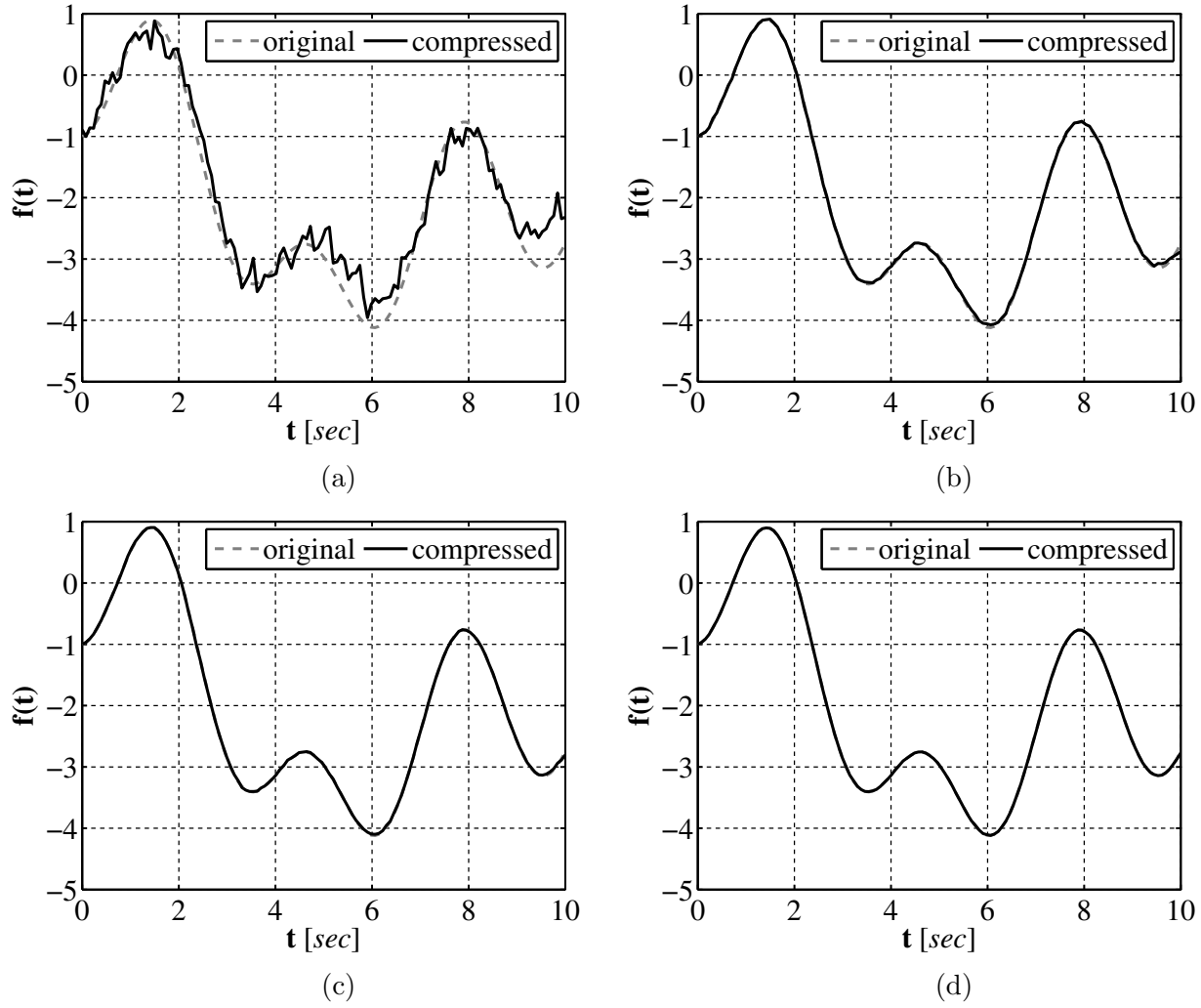


Figure 9.3: The test function reconstructed by solving the compressed sensing convex optimization problem for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data.

9.3.3. Compressed Sensing Embedded Kalman Filtering (CSEKF)

In the previous section a reconstruction approach was introduced that allows for the reconstruction of compressed signals using less data than expected based on the Nyquist/Shannon

sampling principle. As seen in Figure 9.3 this method does a better job at reconstructing a sparse signal than the approach that reconstructs the signal by minimizing the l_2 norm. The authors of [92] introduce an approach that performs the compressed sensing reconstructing using an augmented Kalman Filter (KF) ([74]) which is easy to implement, as opposed to the compressed sensing approach that must solve a convex minimization problem, commonly with an external software package. In order to use the KF it is assumed that the full set of coefficients evolve from time step to time step according to

$$\mathbf{c}_k = \mathbf{A}_k \mathbf{c}_{k-1} + \boldsymbol{\epsilon}_k, \tag{9.32}$$

where $\mathbf{A}_k \in \mathbb{R}^{n \times n}$ is the state transition matrix of the system which describes how the signal coefficients change between time steps, and $\boldsymbol{\epsilon}_k \in \mathbb{R}^n$ is a zero mean Gaussian vector with covariance $\mathbf{R}_k \geq 0$. The compressed form of the coefficients are generated according to

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{c}_k + \boldsymbol{\delta}_k, \tag{9.33}$$

where $\mathbf{H}_k \in \mathbb{R}^{m \times n}$ is the same compression matrix discussed in (9.20) and $\boldsymbol{\delta}_k \in R^m$ is a zero mean Gaussian vector with covariance $\mathbf{Q}_k > 0$.

The CSEKF algorithm first performs a standard KF estimation that is carried out in a two parts, a *prediction* phase and a *correction* phase. The KF estimates $p\left(\mathbf{c}_k \mid \mathbf{y}_k\right)$ as a Gaussian distribution that can be represented using a mean vector $\hat{\mathbf{c}}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. The KF generates the estimate using a standard set of five equations [74]

$$\bar{\hat{\mathbf{c}}}_k = \mathbf{A}_k \hat{\mathbf{c}}_{k-1}, \tag{9.34}$$

$$\bar{\boldsymbol{\Sigma}}_k = \mathbf{A}_k \boldsymbol{\Sigma}_{k-1} \mathbf{A}_k^{\mathrm{T}} + \mathbf{R}_k, \tag{9.35}$$

$$\mathbf{K}_k = \bar{\boldsymbol{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k \bar{\boldsymbol{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}, \tag{9.36}$$

$$\hat{\mathbf{c}}_k = \bar{\hat{\mathbf{c}}}_k + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{H}_k \bar{\hat{\mathbf{c}}}_k \right), \tag{9.37}$$

$$\boldsymbol{\Sigma}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\boldsymbol{\Sigma}}_k, \tag{9.38}$$

where $\mathbf{I}$ in (9.38) is an m dimensional identity matrix. The estimate produced by the KF minimizes the error, using the l_2 norm, between the estimate and the actual set of coefficients

$$\min \|\mathbf{c}_k - \hat{\mathbf{c}}_k\|_2^2 , \tag{9.39}$$

however, this is not what we are trying to solve according to the compressed sensing approach. The CSEKF presented by the authors replaces the classic compressed sensing convex optimization problem (9.31) with the dual problem

$$\min \|\mathbf{c}_k - \hat{\mathbf{c}}_k\|_2^2 \quad \text{subject to} \quad \|\hat{\mathbf{c}}_k\|_1 \leq \epsilon'. \tag{9.40}$$

They solve this constrained optimization problem by iteratively applying the pseudo-observation method presented in [93] to the estimate generated by the KF. According to this approach a pseudo-observation is generated using the constraint being applied to the estimate which is defined as

$$\hat{z}_k = \|\hat{\mathbf{c}}_k\|_1 - \epsilon', \tag{9.41}$$

where the constraining value ϵ' is now treated as a Gaussian noise on the pseudo-measurement with a covariance of $\mathbf{P}$. Using the definition of $\|\cdot\|_1$ the pseudo-measurement can be written as

$$\hat{\mathbf{z}}_k = \overline{\mathbf{H}}_k \hat{\mathbf{c}}_k - \epsilon', \tag{9.42}$$

where the observation matrix of the pseudo-observation is defined as

$$\bar{\mathbf{H}} = \left[\begin{array}{ccc} \text{sign}(\hat{\mathbf{c}}_k(1)) & \cdots & \text{sign}(\hat{\mathbf{c}}_k(n)) \end{array} \right], \tag{9.43}$$

and

$$\text{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{otherwise} \end{cases} . \tag{9.44}$$

The constraint is then iteratively applied τ times to the estimate produced by the KF using the standard KF correction equations (9.36)-(9.38). First, a new Kalman matrix is

generated using the observation matrix of the pseudo-measurement and the covariance of the constraint

$$\overline{\mathbf{K}} = \mathbf{\Sigma}_k \overline{\mathbf{H}}_k^{\mathrm{T}} \left(\overline{\mathbf{H}}_k \mathbf{\Sigma}_k \overline{\mathbf{H}}_k^{\mathrm{T}} + \mathbf{P} \right)^{-1}. \tag{9.45}$$

Next, the mean of the estimate is updated using the standard mean update equation with the assumption that the measured value is 0. Using this assumption the mean update equation can be simplified to

$$\hat{\mathbf{c}}_k = \left(\mathbf{I} - \overline{\mathbf{K}\mathbf{H}} \right) \hat{\mathbf{c}}_k. \tag{9.46}$$

Finally, the covariance of the estimate is constrained using the standard covariance measurement update equation

$$\mathbf{\Sigma}_k = \left(\mathbf{I} - \overline{\mathbf{K}\mathbf{H}}_k \right) \mathbf{\Sigma}_k. \tag{9.47}$$

The full algorithm that combines all of these separate components to reconstruct the compressed signal using the CSEKF is shown in Algorithm 9.3.

The reconstructed test function for the same four test percentages as in the previous two algorithms are shown in Figure 9.4. The algorithm is implemented as shown in Algorithm 9.3 with the tuning parameters being chosen as: $\mathbf{Q}_k = 0.01^2$, $\mathbf{R}_k = 0$, $\mathbf{P} = 100^2$, and $\tau = 100$. Since the input vector is static the initial estimate of the coefficient vector was zero, $\hat{\mathbf{c}}_{k-1} = 0^n$, the state transition matrix is chosen as $\mathbf{A}_k = \mathbf{I}^n$, and the initial covariance matrix of the estimate is chosen to be $\mathbf{\Sigma}_{k-1} = \mathbf{I}^n$.

9.3.4. Kalman Filter Constrained with Quasi-Norm (CSEKF-p)

In the previous section an approach was presented that solves a problem similar to the standard compressed sensing problem (9.31) using a KF that is constrained by iteratively applying a pseudo-observation to the estimate generated by the KF. The standard approach to solving the compressed sensing problem replaces $\|\cdot\|_0$ in (9.29) with $\|\cdot\|_1$. A new approach has been presented by the authors of [49] that replaces the zero norm with $\|\cdot\|_p$ where

Algorithm 9.3 Overview of the CSEKF reconstruction algorithm.

procedure CSEKF($\hat{\mathbf{c}}_{k-1}, \boldsymbol{\Sigma}_{k-1}$)
    Data:
        $\hat{\mathbf{c}}_{k-1}$ the estimate from the previous time step
        $\boldsymbol{\Sigma}_{k-1}$ the covariance matrix from the previous time step

    Result:
        $\hat{\mathbf{c}}_k$ the estimate from the current time step
        $\boldsymbol{\Sigma}_k$ the covariance matrix from the current time step

    $\hat{\mathbf{c}}_k \leftarrow \mathbf{A}_k \hat{\mathbf{c}}_{k-1}$
    $\boldsymbol{\Sigma}_k \leftarrow \mathbf{A}_k \boldsymbol{\Sigma}_{k-1} \mathbf{A}_k^{\mathrm{T}} + \mathbf{R}_k$
    $\mathbf{K} = \boldsymbol{\Sigma}_k \mathbf{H}^{\mathrm{T}} \left(\mathbf{H}_k \boldsymbol{\Sigma}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}$
    $\hat{\mathbf{c}}_k + \mathbf{K} \left(\mathbf{y} - \mathbf{H}_k \hat{\mathbf{c}}_k \right)$
    for $i = 1, \tau - 1$ **do**
        $\overline{\mathbf{H}} = \left[\begin{array}{ccc} \mathrm{sign}(\hat{\mathbf{c}}_k(1)) & \cdots & \hat{\mathbf{c}}_k(n) \end{array} \right]$
        $\overline{\mathbf{K}} = \boldsymbol{\Sigma}_k \overline{\mathbf{H}}^{\mathrm{T}} \left(\overline{\mathbf{H}} \boldsymbol{\Sigma}_k \overline{\mathbf{H}}^{\mathrm{T}} + \mathbf{P} \right)^{-1}$
        $\hat{\mathbf{c}}_k = \left(\mathbf{I} - \overline{\mathbf{K}} \overline{\mathbf{H}} \right) \hat{\mathbf{c}}_k$
        $\boldsymbol{\Sigma}_k = \left(\mathbf{I} - \overline{\mathbf{K}} \overline{\mathbf{H}} \right) \boldsymbol{\Sigma}_k$
    end for
end procedure

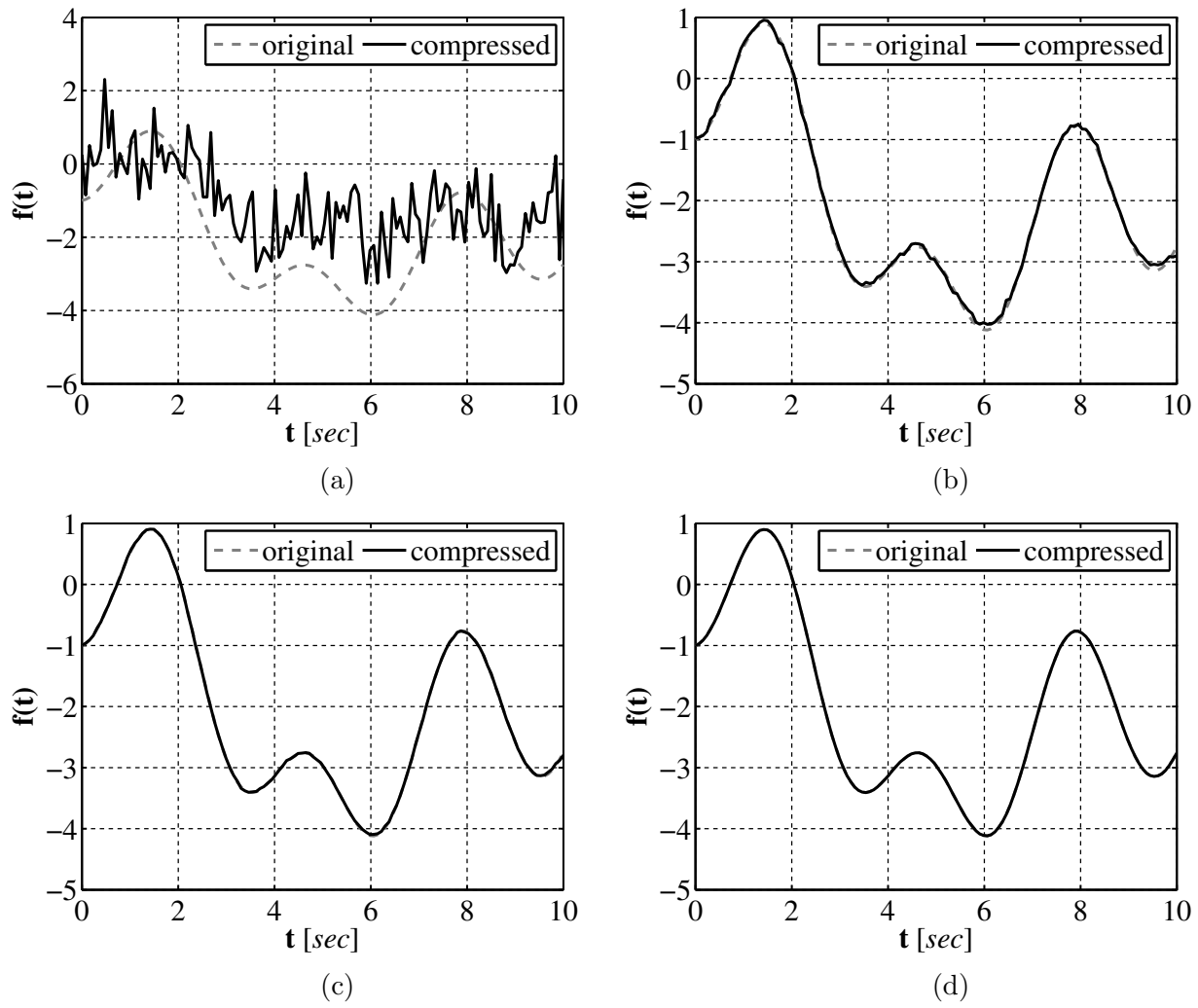


Figure 9.4: The test function reconstructed using the compressed sensing embedded Kalman Filter for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data.

$0 < p < 1$, which has been shown to yield better accuracy than using $\|\cdot\|_1$ in some cases. Using this approach, we would like to constrain the estimate produced by the KF with the quasi-norm $\|\cdot\|_p$ as opposed to $\|\cdot\|_1$.

Just as in the previous section the initial estimate is produced using the standard KF equations. The pseudo-observation used to apply the constraint to the KF estimate using $\|\cdot\|_p$ is defined as

$$\hat{\mathbf{z}}_k = \|\hat{\mathbf{c}}_k\|_p - \epsilon', \tag{9.48}$$

where ϵ' is again treated as noise on the pseudo-observation with a covariance of $\mathbf{P}$ and where the quasi-norm is defined as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |\mathbf{x}(i)|^p \right)^{1/p}. \tag{9.49}$$

Unlike in the previous case where the constraint was linear and could be applied using the pseudo-observation matrix, the constraint applied by the p-norm is nonlinear and can be rewritten as

$$\hat{\mathbf{z}}_k = \mathbf{h}\left(\hat{\mathbf{c}}_k\right) - \epsilon', \tag{9.50}$$

where

$$\mathbf{h}(\hat{\mathbf{c}}_k) = \|\hat{\mathbf{c}}_k\|_p. \tag{9.51}$$

Since the constraint using the p-norm is nonlinear the constraint is applied to the KF estimate by iteratively performing a measurement update using (9.50) and the Extended Kalman Filter (EKF) ([94]) update equations. Just as in the linear case the first step in applying the constraint is to construct the Kalman matrix according to

$$\overline{\mathbf{K}} = \mathbf{\Sigma}_k \overline{\mathbf{H}}_k^{\mathrm{T}} \left(\overline{\mathbf{H}}_k \mathbf{\Sigma}_k \overline{\mathbf{H}}_k^{\mathrm{T}} + \mathbf{P} \right)^{-1}, \tag{9.52}$$

where $\overline{\mathbf{H}}$ is now the Jacobian of $\mathbf{h}(\cdot)$ and is defined as

$$\overline{\mathbf{H}}(i) = \begin{cases} \left(\sum_{i=1}^{n} |\hat{\mathbf{c}}_k^*(i)|^p\right)^{1/(p-1)} |\hat{\mathbf{c}}_k^*(i)|^{p-1} & \hat{\mathbf{c}}_k^*(i) \geq 0, \\ -\left(\sum_{i=1}^{n} |\hat{\mathbf{c}}_k^*(i)|^p\right)^{1/(p-1)} |\hat{\mathbf{c}}_k^*(i)|^{p-1} & \hat{\mathbf{c}}_k^*(i) < 0, \end{cases}. \tag{9.53}$$

where $\hat{\mathbf{c}}_k^* = \hat{\mathbf{c}}_k$. The constraint is applied to the mean of the estimate using the standard EKF mean measurement update equation, again with the assumption that the measured value is zero, which when simplified is written as

$$\hat{\mathbf{c}}_k = \hat{\mathbf{c}}_k - \overline{\mathbf{K}} \|\hat{\mathbf{c}}_k\|_p . \tag{9.54}$$

The final step in applying the constraint is to update the covariance of the estimate using the standard EKF covariance measurement update equation

$$\boldsymbol{\Sigma}_k = \left(\mathbf{I} - \overline{\mathbf{K}\mathbf{H}_k}\right) \boldsymbol{\Sigma}_k. \tag{9.55}$$

The complete algorithm for reconstructing a compressed signal using the estimate produced by the KF constrained using the p-norm is shown in Algorithm 9.4.

The reconstruction results for the Compressed Sensing Kalman Filter using a quasi-norm to constrain the estimate are presented in Figure 9.5 for the same four data size used for the three previous algorithms. A parameter that greatly effects the performance of the algorithm is the value chosen for p. To determine the best value of p the algorithm was run for each data percentage for 20 equally spaced values of $p \in (0, 1)$ and the value of p that corresponded to the smallest reconstruction error was selected. A plot of the mean square error (MSE) between the compressed signal and original signal can be seen in Figure 9.6 for the a compressed signal stored using 75% of the original data. The remainder of the tuning parameters for the algorithm were selected to be the same as those used for the CSEKF reconstruction.

9.3.5. Algorithm Comparison

Before deciding which reconstruction algorithm should be used in the FastSLAM COG algorithm we examine the performance of each of the algorithms in two key areas: accuracy and run-time. First, we examine how well each algorithm reproduces the original signal

Algorithm 9.4 Overview of CSEKF-p reconstruction algorithm.

procedure CSEKFP($\hat{\mathbf{c}}_{k-1}, \mathbf{\Sigma}_{k-1}$)
    Data:
        $\hat{\mathbf{c}}_{k-1}$ the estimate from the previous time step
        $\mathbf{\Sigma}_{k-1}$ the covariance matrix from the previous time step

    Result:
        $\hat{\mathbf{c}}_k$ the estimate from the current time step
        $\mathbf{\Sigma}_k$ the covariance matrix from the current time step

    $\hat{\mathbf{c}}_k \leftarrow \mathbf{A}_k \hat{\mathbf{c}}_{k-1}$
    $\mathbf{\Sigma}_k \leftarrow \mathbf{A}_k \mathbf{\Sigma}_{k-1} \mathbf{A}_k^{\mathrm{T}} + \mathbf{R}_k$
    $\mathbf{K} = \mathbf{\Sigma}_k \mathbf{H}^{\mathrm{T}} \left(\mathbf{H}_k \mathbf{\Sigma}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}$
    $\hat{\mathbf{c}}_k + \mathbf{K} \left(\mathbf{y} - \mathbf{H}_k \hat{\mathbf{c}}_k \right)$
    for $i = 1, \tau - 1$ **do**
        for $j = 1, n$ **do**
            if $\hat{\mathbf{c}}_k(j) \geq 0$ **then**
                $\overline{\mathbf{H}}(i) = \left(\sum_{i=1}^{n} |\hat{\mathbf{c}}_k^*(i)|^p \right)^{1/(p-1)} |\hat{\mathbf{c}}_k^*(i)|^{p-1}$
            else
                $\overline{\mathbf{H}}(i) = - \left(\sum_{i=1}^{n} |\hat{\mathbf{c}}_k^*(i)|^p \right)^{1/(p-1)} |\hat{\mathbf{c}}_k^*(i)|^{p-1}$
            end if
        end for
        $\overline{\mathbf{H}} = \begin{bmatrix} \operatorname{sign}(\hat{\mathbf{c}}_k(1) & \cdots & \hat{\mathbf{c}}_k(n) \end{bmatrix}$
        $\overline{\mathbf{K}} = \mathbf{\Sigma}_k \overline{\mathbf{H}}^{\mathrm{T}} \left(\overline{\mathbf{H}} \mathbf{\Sigma}_k \overline{\mathbf{H}}^{\mathrm{T}} + \mathbf{P} \right)^{-1}$
        $\hat{\mathbf{c}}_k = \left(\mathbf{I} - \overline{\mathbf{K}} \overline{\mathbf{H}} \right) \hat{\mathbf{c}}_k$
        $\mathbf{\Sigma}_k = \left(\mathbf{I} - \overline{\mathbf{K}} \overline{\mathbf{H}} \right) \mathbf{\Sigma}_k$
    end for
end procedure

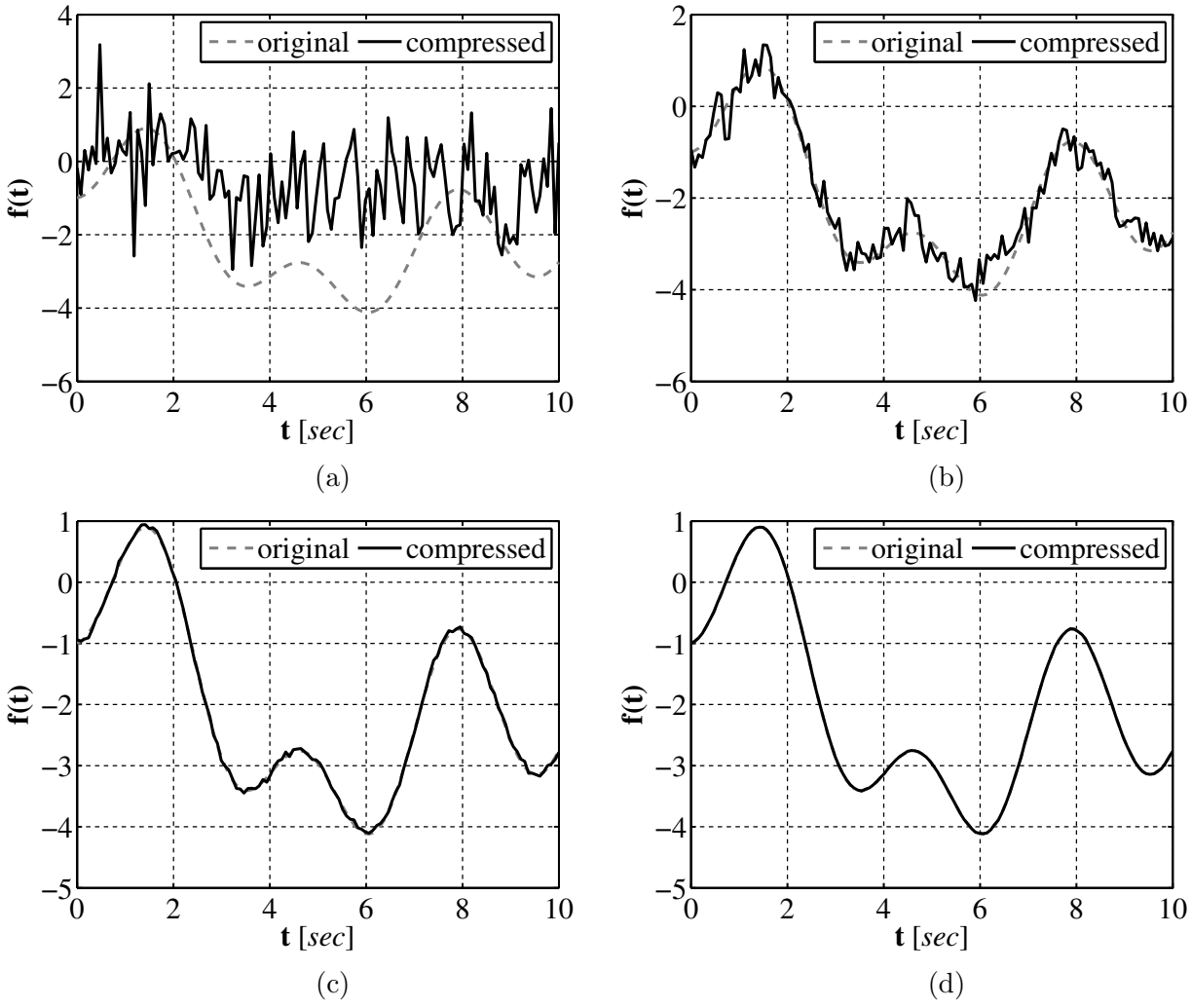


Figure 9.5: The test function reconstructed using the quasi-norm constrained Kalman Filter for compressed size of (a) 25%, (b) 50%, (c) 75%, and (d) 95% of the original data.

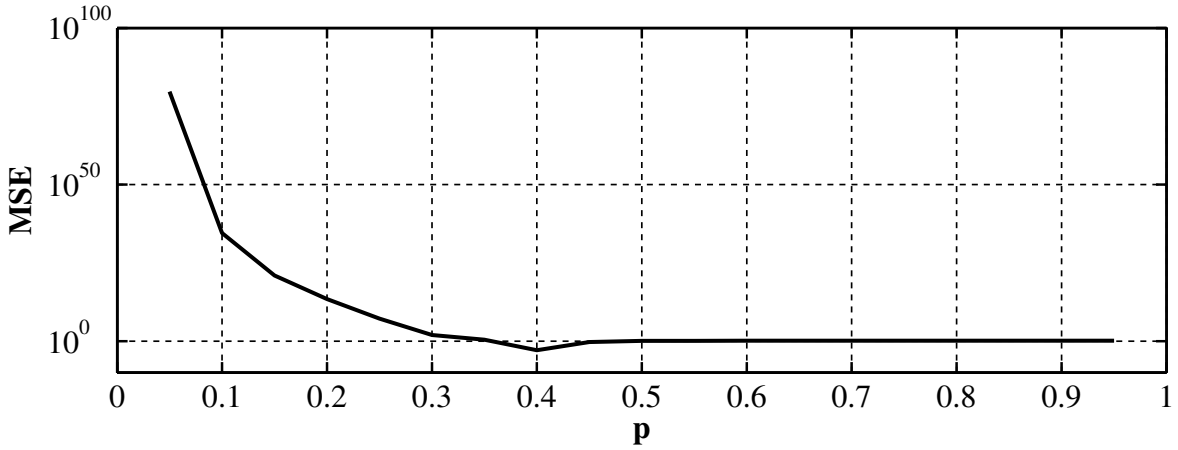


Figure 9.6: MSE for different values of p for a single compressed percentage 75%.

based on the amount of information that is stored in the compressed signal. To compare how well each algorithm reconstructs the signal, each algorithm was run for 20 equally spaced percentages in $(0\%, 100\%]$. The results for each of the algorithms are shown in Figure 9.7 which plots the MSE as a function of the amount of data stored. As seen in the results, in general each of the compressed sensing type approaches produce less of an error for a given data percentage than the l_2 approach.

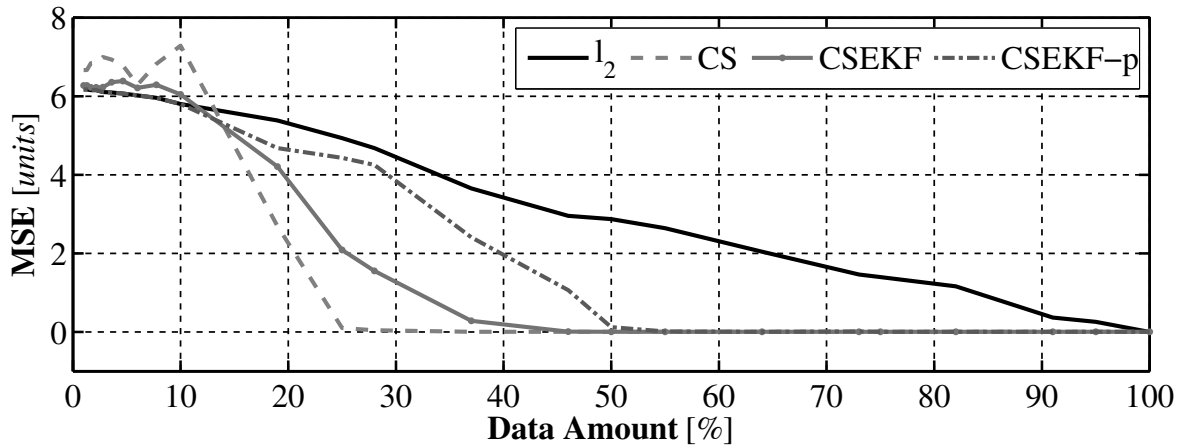


Figure 9.7: MSE as a function of signal data used for each of the four reconstruction algorithms.

The second key characteristic of each algorithm examined is the time it takes for each

algorithm to run. A process similar to the error performance testing described above was used to generate the timing results. In order to better illustrate the timing results the same test function was used, (9.19), however this time evaluated at 1024 evenly spaced locations in $[0, 10]$. The run time as a function of the amount of compressed data used to represent the signal can be seen in Figure 9.8. As seen in the timing results the two algorithms that use the augmented Kalman filter have run times that are significantly larger than the other approaches. This increased run time is due to the number of matrix-matrix multiplications required by the Kalman filter which are computationally heavy, from [95] having $\mathcal{O}(n^{\log_2 7})$. The remaining two algorithms, l_2 and compressed sensing, are implementing using matrix-vector multiplications which can be performed more quickly than matrix-matrix multiplication.

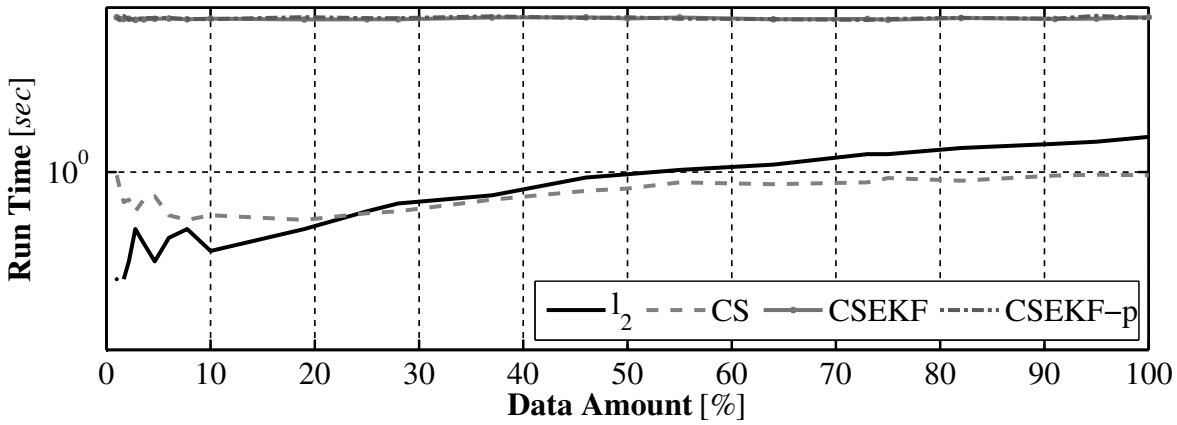


Figure 9.8: Run-time as a function of compressed percentage for each of the four reconstruction algorithms.

9.4. Performance Improvements

Our overall goal is to augment the FastSLAM algorithm to use compressed occupancy grids. In order for us to accomplish this task the reconstruction approach selected must run thousands or hundreds of thousands of times during the operation of a UMV. By using a Rao-

Blackwellized particle filter, as the number of particles increases the higher the probability that the SLAM posterior estimated by the algorithm approximates the true SLAM posterior. As seen in the Algorithm 9.1 the reconstruction process must be performed by each particle at each iteration, thus the execution time of the reconstruction process is important in the overall performance of the algorithm.

In Section 9.3 four different algorithms were presented that can be used to reconstruct an occupancy grid from the compressed form. From the timing results (Figure 9.8), the two approaches that make use of the Kalman filter (CSEKF and CSEKF-p) take much longer to run than l_2 and CS. The longer run times of the Kalman filter based approaches are due to the large number of matrix-matrix multiplications. Based on their performance the CSEKF and CSEKF-p algorithms will not be used in FastSLAM COG. We will now present a method that can be used to improve the accuracy of the remaining two algorithms and a method that improves the run time of the l_2 approach.

9.4.1. Updated Compression Step

As seen in Section 9.2.1 the compressed occupancy grid is generated by selecting a subset of the information stored in the coefficients. This compression is performed according to (9.4) which makes use of $\mathbf{H}$ to perform the compression. As stated in Section 9.3.2 we would like for this compression matrix to obey the RIP property. Several common matrices that are used in compressed sensing applications are the discrete Fourier transform ([96]), the discrete cosine transform ([86]), the Hadamard transform ([97]), and noiselet transform ([98]). Each of these matrices are constructed in such a way that each of the elements stored in the compressed vector stores a small amount of information about each of the coefficients used to represent the original signal and this information can be recovered by knowing $\mathbf{H}$. These types of matrices are useful when there is no knowledge about the sparse signal that is being

compressed, however we are constructing the sparse signal thus we have full knowledge of the signal before the compression is performed. Using this knowledge we present an alternative form of $\mathbf{H}$ that allows for better reconstruction using a smaller amount of information.

The form of $\mathbf{H}$ selected comes from the definition of the square error defined according to the l_2 norm. For some discrete signal $\mathbf{x} \in \mathbb{R}^n$ the square of the error between the signal and the approximation of the signal $\hat{\mathbf{x}}$ generated using $m < n$ elements is defined as

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \langle \mathbf{x} - \hat{\mathbf{x}} | \mathbf{x} - \hat{\mathbf{x}} \rangle \,, \tag{9.56}$$

where $\langle \cdot | \cdot \rangle$ is the standard inner product. The discrete signal $\mathbf{x} \in \mathbb{R}^n$ can be represented in some alternate basis so that a single element can be expressed as

$$\mathbf{x}(i) = \sum_{j=1}^{n} \mathbf{\Phi}\,(i,j)\,\mathbf{c}(j). \tag{9.57}$$

A single element of the compressed form of the signal can also be expressed in the alternate basis according to

$$\hat{\mathbf{x}}(i) = \sum_{j=1}^{m} \hat{\mathbf{\Phi}}\,(i,j)\,\hat{\mathbf{c}}(j). \tag{9.58}$$

For some arbitrary compression matrix, the matrix of basis functions used to reconstruct the compressed form of the signal is defined as

$$\hat{\mathbf{\Phi}} = \mathbf{\Phi}\mathbf{H}^{\dagger}, \tag{9.59}$$

and the set of coefficients used to reconstruct the discrete signal are

$$\hat{\mathbf{c}} = \mathbf{y}. \tag{9.60}$$

While this is a valid approach for compressing a signal, we would like to simplify the compression process for performance reasons. Instead of generating a new set of basis function it is easier to keep the basis functions the same at all times and just select a subset of

the coefficients to represent the signal. In order to decide which coefficients should be selected that minimize the error based on the l_2 norm, we define several sets. The set $\boldsymbol{\zeta} = \{x | x \in \mathbb{N}, 1 \leq x \leq n\}$ is the full set of indices of the discrete vector being compressed. The set $\boldsymbol{\lambda} \in \mathbb{N}^m$ is the set of indices that are used to represent the compressed signal, $\boldsymbol{\lambda} \subset \boldsymbol{\zeta}$. Finally, the set $\boldsymbol{\gamma} \in \mathbb{N}^{n-m}$ is the set of indices of coefficients not used in the compressed signal, $\boldsymbol{\gamma} = \{x | x \in \zeta \text{ and } x \notin \boldsymbol{\lambda}\}$. Using these sets and the fact that a constant basis is used to represent the uncompressed and compressed signal an element in the reconstructed signal can be written as

$$\hat{\mathbf{x}}(i) = \sum_{j=1}^{m} \boldsymbol{\Phi}(i, \boldsymbol{\lambda}(j)) \mathbf{c}(\boldsymbol{\lambda}(j)). \tag{9.61}$$

The square of the error in the l_2 norm for a single element becomes

$$\begin{aligned}
\|\mathbf{x}(i) - \hat{\mathbf{x}}(i)\|_2^2 &= \langle \mathbf{x}(i) - \hat{\mathbf{x}}(i) | \mathbf{x}(i) - \hat{\mathbf{x}}(i) \rangle , && (9.62)\\
&= \left\langle \sum_{j=1}^{n-m} \boldsymbol{\Phi}(i, \boldsymbol{\gamma}(j)) \mathbf{c}(\boldsymbol{\gamma}(j)) \middle| \sum_{k=1}^{n-m} \boldsymbol{\Phi}(i, \boldsymbol{\gamma}(k)) \mathbf{c}(\boldsymbol{\gamma}(k)) \right\rangle , && (9.63)\\
&= \sum_{j=1}^{n-m} \sum_{k=1}^{n-m} \mathbf{c}(\boldsymbol{\gamma}(j)) \mathbf{c}(\boldsymbol{\gamma}(k)) \langle \boldsymbol{\Phi}(i, \boldsymbol{\gamma}(j)) | \boldsymbol{\Phi}(i, \boldsymbol{\gamma}(k)) \rangle , && (9.64)\\
&= \sum_{j=1}^{n-m} \boldsymbol{\Phi}(i, \boldsymbol{\gamma}(j))^2 \mathbf{c}(\boldsymbol{\gamma}(j))^2, && (9.65)
\end{aligned}$$

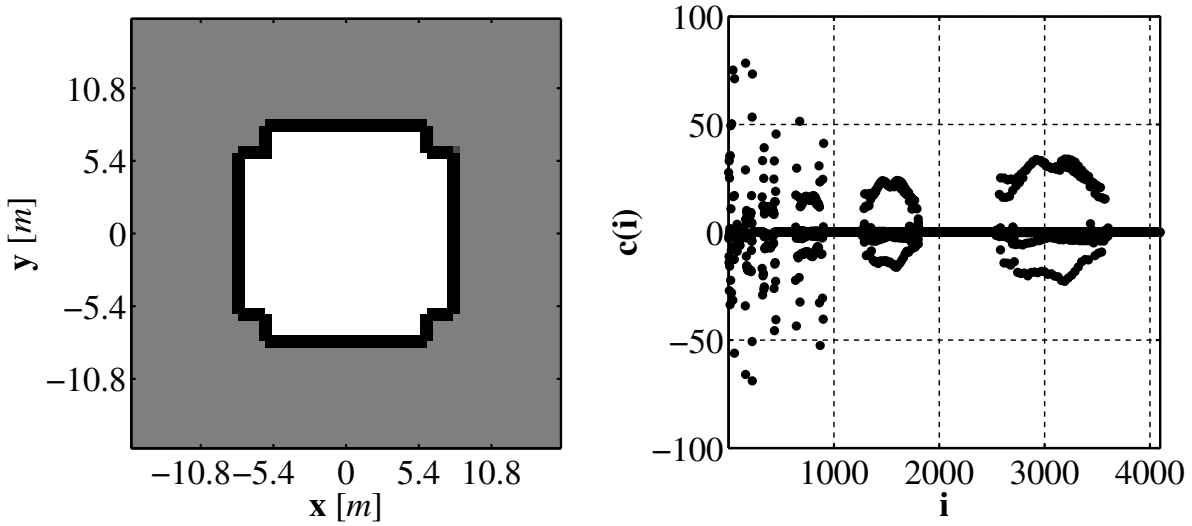
and if we assume that the set of basis function is orthonormal, which is common, then the l_2 error for the entire compressed signal becomes

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{n-m} \mathbf{c}(\boldsymbol{\gamma}(j))^2. \tag{9.66}$$

This tells us that the best way to minimize the error in the reconstructed signal is to select the coefficients that have the largest magnitudes. Using this knowledge a new compression algorithm that does not require matrix-vector multiplications and selects only the m coefficients with the largest magnitudes was developed and shown in Algorithm 9.5.

Algorithm 9.5 Overview of the updated compression algorithm that selects the m coefficients with largest magnitudes.

```
1: procedure FastCompression(c, m)
2:     Data:
3:         c the vector of coefficients
4:         m the desired size of the compressed signal
5:
6:     Result:
7:         y the compressed signal
8:
9:     x = sort(abs(c),'descend')
10:    for i = 1, m do
11:        i(i) = x(i)
12:        y(i) = c(x(i))
13:    end for
14: end procedure
```



(a) (b)

Figure 9.9: An example occupancy grid of an unknown environment (a) and the coefficients that represent it using Haar wavelet basis (b).

To examine how the use of the new compression algorithms affects the performance of the reconstruction algorithms, the occupancy grid shown in Figure 9.9a is reconstructed from the compressed form. The compression matrix selected for the baseline results is different from the orthonormal gaussian random matrix used in Section 9.3, in its place a normalized Hadamard matrix was used. Each of the two algorithms were then run using each of the two compression methods for 20 separate compression percentages in $(0, 100\%)$. The performance results of each of the two algorithms are provided in Figure 9.10. As seen from the results for each of the two reconstruction algorithms the occupancy grid is reconstructed with a smaller MSE using the compression matrix that keeps the m most significant coefficients as opposed to the compression step using the normalized Hadamard matrix using the same amount of data.

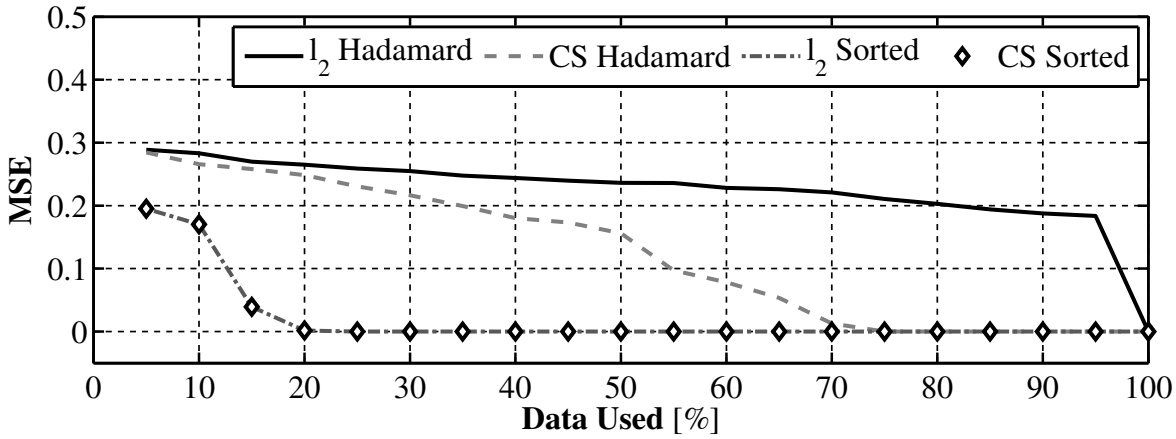


Figure 9.10: Comparison of the reconstruction performance of the normalized Hadamard compression matrix and the matrix that selects the m most significant coefficients.

9.4.2. Updated l_2 reconstruction

In Section 9.3.1 a method for reconstructing a compressed signal was introduced that minimizes the error according to the l_2 norm. The method makes uses of the Moore-Penrose

pseudo inverse to reconstruct the set of coefficients from their compressed form according to

$$\hat{\mathbf{c}} = \mathbf{H}^{\dagger}\mathbf{y}. \tag{9.67}$$

This method of reconstructing the signal is made up of two steps where the performance of each step is based on the size of the signal (n). The first step in the process is the calculation of the pseudoinverse of the compression matrix $\mathbf{H}$. Many common software implementations of this calculation, for example Matlab's pinv() function ([99]), makes use of the Singular Value Decomposition (SVD) of the matrix. The process of calculating the SVD of a matrix is worse from a performance perspective than that of matrix-matrix multiplication which is $\mathcal{O}(mn^2)$ ([100]) for a $m \times n$ matrix.

The second step of the algorithm is a matrix-vector multiplication operation which in a naive implementation has a performance of $\mathcal{O}(n^2)$. We would like to improve the performance of the reconstruction algorithm to get rid of these computationally intensive steps. By making use of the compression matrix developed in the previous section that just selects the m coefficients with the largest magnitudes, the performance of the reconstruction process can be improved. Each of the rows in the compression matrix $\mathbf{H}$ are orthogonal, that is for two rows in the selection matrix $\mathbf{H}(i,:)$ and $\mathbf{H}(j,:)$ the discrete inner product is zero, $\langle \mathbf{H}(i,:) | \mathbf{H}(j,:) \rangle = 0$. A second property is that each row of $\mathbf{H}$ is normalized, $\sum_{j=1}^{n} \mathbf{H}\,(i,j) = 1$, thus $\mathbf{H}$ is orthonormal. The pseudoinverse of a orthonormal matrix can be easily computed without the computationally intensive SVD process according to

$$\mathbf{H}^{\dagger} = \mathbf{H}^{*}, \tag{9.68}$$

where $\mathbf{H}^{*}$ is the conjugate transpose of $\mathbf{H}$ and is found by taking the transpose of $\mathbf{H}$ followed by the complex conjugate of each element in $\mathbf{H}$. Since the compression matrix is composed of elements containing just 0 or 1 the pseudoinverse is just the transpose $\mathbf{H}^{\dagger} = \mathbf{H}^{\mathrm{T}}$.

If we return to (9.67) we can replace the pseudoinverse of $\mathbf{H}$ with the transpose thus, the

reconstruction process becomes

$$\hat{\mathbf{c}} = \mathbf{H}^{\mathrm{T}}\mathbf{y}. \tag{9.69}$$

We can improve the performance even more if we assume that the compression is performed as discussed above, that is where the m coefficients with the largest magnitudes are selected. In that case then $\mathbf{H}$ is full of zeros except for elements that correspond to the coefficients with largest magnitudes. Using this information we simplify the l_2 reconstruction algorithm to just generate a vector of zeros and then copy the stored coefficients into their proper locations in the reconstructed vector. This replaces the matrix-vector multiplication with m value copies. The updated form of the l_2 reconstruction algorithm that does not require the potentially expensive calculation of the matrix pseudo-inverse or matrix-vector multiplication is presented in Algorithm 9.6.

Algorithm 9.6 Overview of the optimized l_2 reconstruction algorithm.

```
 1: procedure FastLTWOReconstruction(y,n)
 2:     Data:
 3:         y the compressed form of the signal, of length m
 4:         n the size of the uncompressed signal
 5:
 6:     Result:
 7:         c the uncompressed signal
 8:
 9:     for i = 1, n do
10:         c(i) = 0
11:     end for
12:     for i = 1, m do
13:         c(i(i)) = y(i)
14:     end for
15: end procedure
```

To examine the performance benefit of the new l_2 reconstruction algorithm a compressed form of the occupancy grid seen in Figure 9.11 is reconstructed using the original algorithm explained in Section 9.3.1 and the new algorithm described in Algorithm 9.6. The algorithm

is run for x varying compressed data sizes in $[0\%, 100\%]$ and the timing results are seen in Figure 9.11. As seen from the results the run time for the new l_2 reconstruction remains small as the size of the compressed signal grows while the original l_2 reconstruction approach slows down as the size of the compressed signal grows.

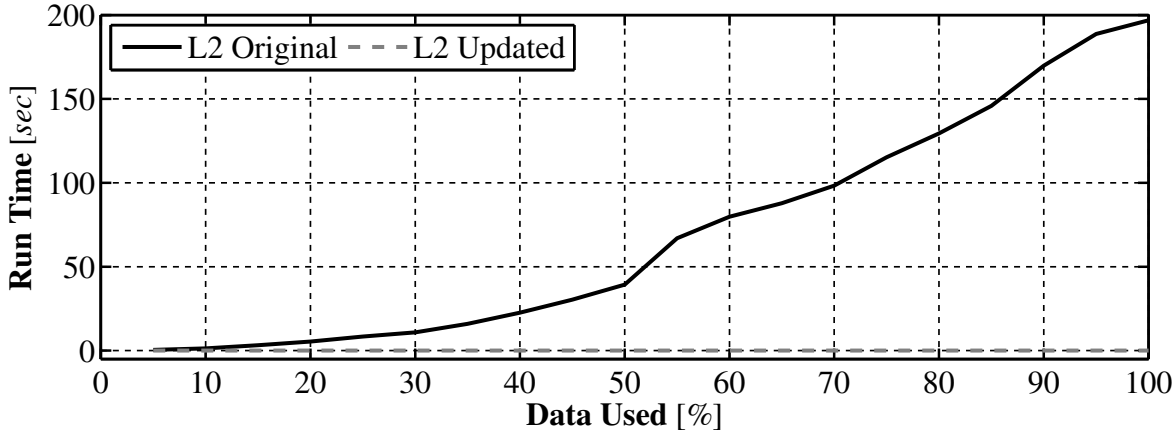


Figure 9.11: The comparison in run time for the original l_2 reconstruction along with the updated form of the reconstruction algorithm.

9.5. Experimental Results

We would now like to examine how using compressed occupancy grids affects the FastSLAM OG algorithm. Based on the timing results presented in Section 9.3 only two of the uncompression methods, the l_2 (Section 9.3.1) and CS (Section 9.3.2), were selected to use in the FastSLAM COG algorithm. Both of the algorithms are compared using data captured by a small unmanned ground vehicle (UGV). The UGV is equipped with Mecanum wheels ([83]) which allow the vehicle to move in almost any direction without the constraints that are typical with many ground vehicles. The use of Mecanum wheels makes it difficult to use standard wheel encoders for localization purposes; to overcome this a downward facing camera is attached to the UGV which provides visual odometry data for localization using the approach described in Chapter 4. Other sensors attached to the UGV include a digi-

tal compass to provide the global heading of the vehicle, a LiDAR sensor to provide range and bearing measurements to objects in the environment, and the custom designed laser rangefinder presented in Chapter 3. As with the FastSLAM AOG algorithm we will first provided results generated using the attached LiDAR then present similar results generated using the custom designed laser based rangefinder.

9.5.1. Results Using LiDAR

For comparison purposes, the standard FastSLAM OG algorithm described in Chapter 7 was used to generate a baseline vehicle path and occupancy grid map. Once the baseline was generated the FastSLAM COG algorithm described in Section 9.2 was run using the same data used by the FastSLAM OG approach. For each of the two reconstruction algorithms the compression method used is the optimized approach described in Section 9.4.1 and the l_2 reconstruction is implemented as described in Section 9.4.2. For the CS reconstruction algorithm the convex optimization problem is solved using the NESTA library.

The UGV was remotely driven around a small enclosed 3m $\times$12m environment while all of the sensor data was logged to memory on a small attached computer. The data was then post processed to produce a baseline vehicle path and an occupancy grid representing the testing environment. The baseline vehicle path, along with the true vehicle path, can be seen in Figure 9.12a and the occupancy grid, with a $0.1m$ cell size is shown in Figure 9.12b.

The vehicle path estimate generated by FastSLAM COG algorithm using l_2 reconstruction can be seen in Figure 9.13a and the reconstructed occupancy grid is shown in Figure 9.13b. The generated path estimate using compressed sensing approach is shown in Figure 9.14a and the reconstructed occupancy grid is shown in Figure 9.14b. For each of the algorithms 40% of the amount of data used to represent the full occupancy grid is used. Figure 9.15 shows the mean square error (MSE) in the path estimate and reconstructed occupancy

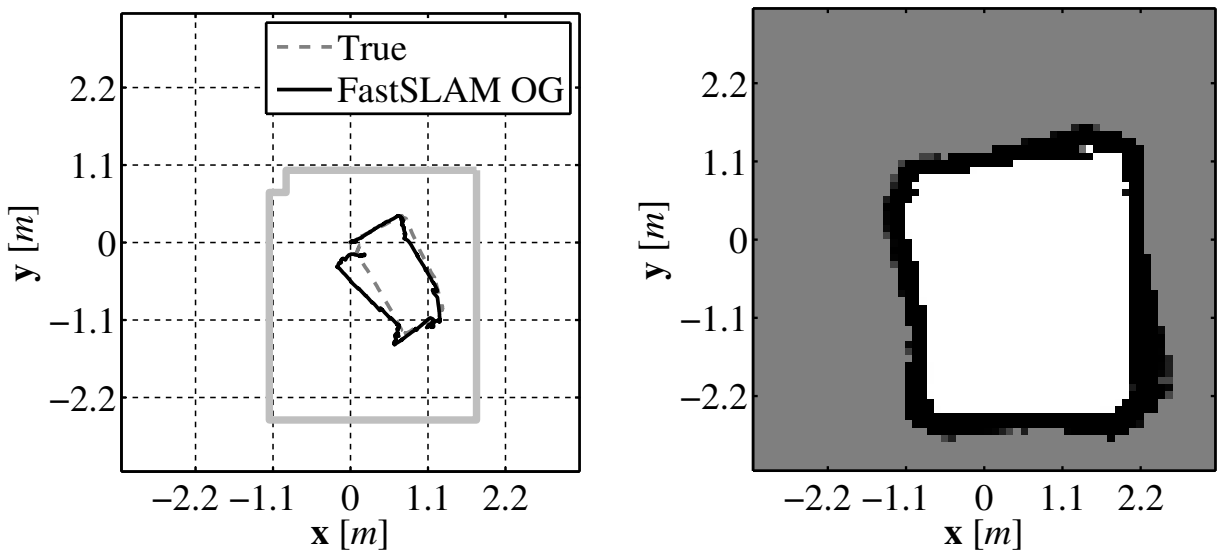
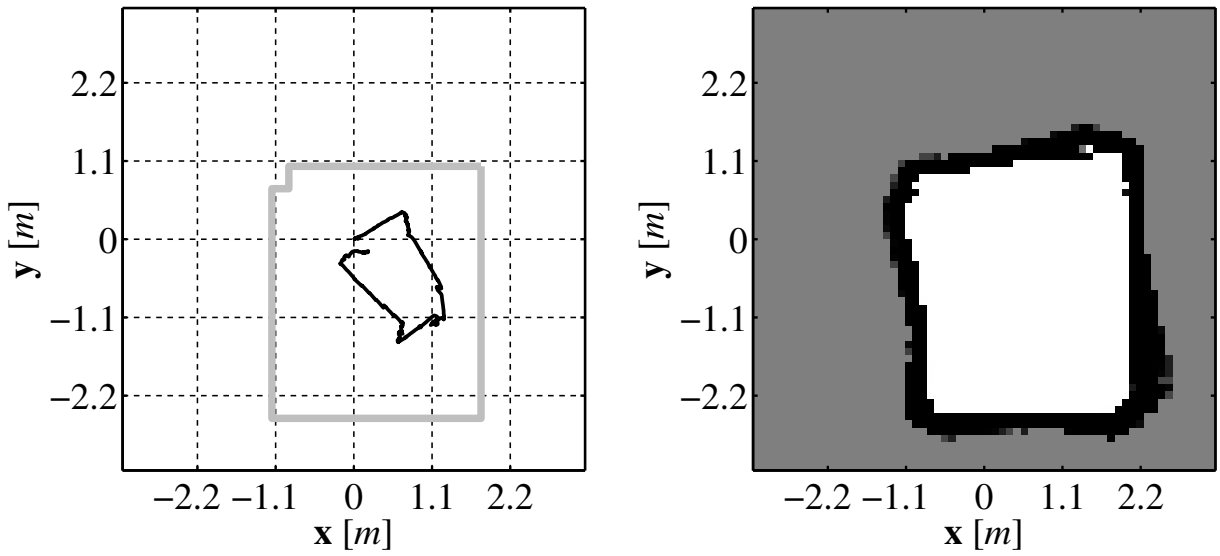


Figure 9.12: The FastSLAM OG produced path estimate (a) and map (b) used as a baseline generated with LiDAR.

grid between the uncompressed form of the algorithm and the compressed form of the algorithm. As Figure 9.15 shows using more than 40% of the data does not decrease the MSE any more than the error that exists when using 40% of the data.

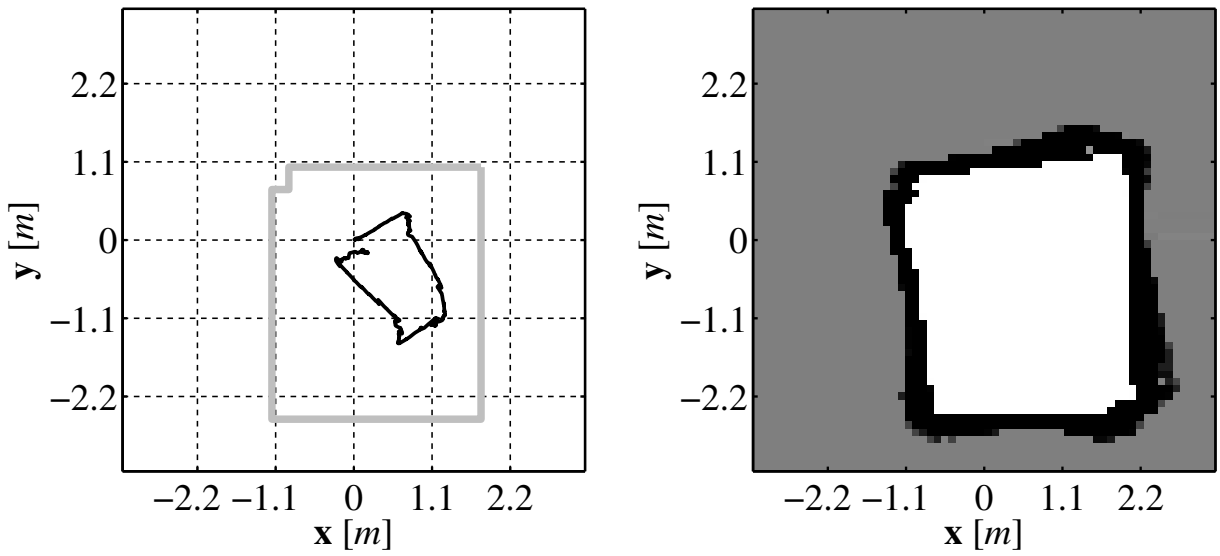
To illustrate how the amount of data used affects the quality of the reconstructed occupancy grid examine the plots provided in Figure 9.16. This set of plots shows a series of occupancy grids that were generated as the final map of by the FastSLAM COG algorithm using the l_2 reconstruction approach for decreasing amounts of data. As can be seen the first several grids, up to 40% of the data show no decrease in quality, however as the amount of data used continues to decrease the amount of error in the final occupancy grid increases until the grid becomes unusable. A similar set of plots are provided in Figure 9.17 for the FastSLAM COG algorithm using the compressed sensing reconstruction approach and similar results can be seen.

By examining Figure 9.15 it can be seen that there is a constant error that appears in



(a) (b)

Figure 9.13: The FastSLAM COG produced path estimate (a) and map (b) generated using the l_2 reconstruction approach with 40% of the data and LiDAR.



(a) (b)

Figure 9.14: The FastSLAM COG produced path estimate (a) and map (b) generated using the compressed sensing reconstruction approach with 40% of the data and LiDAR.

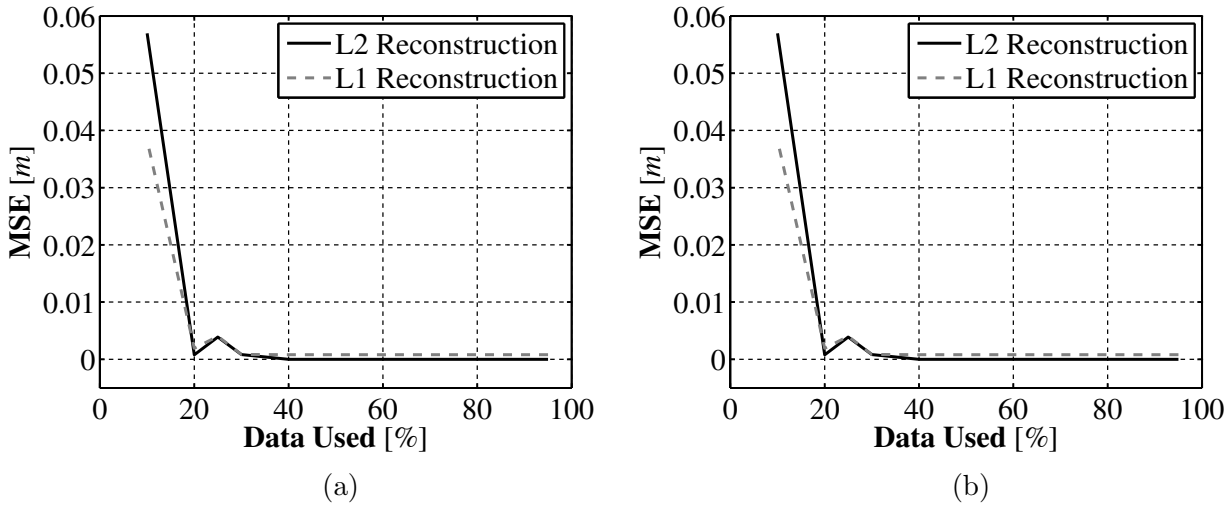


Figure 9.15: The MSE as a function of data percentage for the estimated path (a) and map (b) generated with LiDAR.

both the path estimate and the occupancy grid reconstruction from using the compressed sensing approach. This error comes from the parameters that are chosen for the NESTA library. Of most significance is the value of ϵ from (9.31) which is the amount of error that is allowed between the coefficients that represent the compressed form of the occupancy grid and the value of those coefficients in the reconstructed occupancy grid. For the presented results the value of ϵ was chosen to be 10^{-10} which is quite small, however it does allow for small errors to occur in the reconstructed occupancy grid. These errors can be reduced, or removed completely, from the estimate by choosing a smaller value of ϵ and adjusting the parameters used by the NESTA library. However, as previously discussed the accuracy of the grid reconstruction is not the only property of importance when selected which approach should be used. The other significant property of each algorithm that must be investigated is the time it takes to execute the algorithm. Shown in Figure 9.18 is a comparison of the average time it takes to run a single iteration of the FastSLAM COG algorithm using the l_2 approach and the compressed sensing approach for varying amounts of data. As seen in these results the average run time of the FastSLAM COG algorithm using the compressed

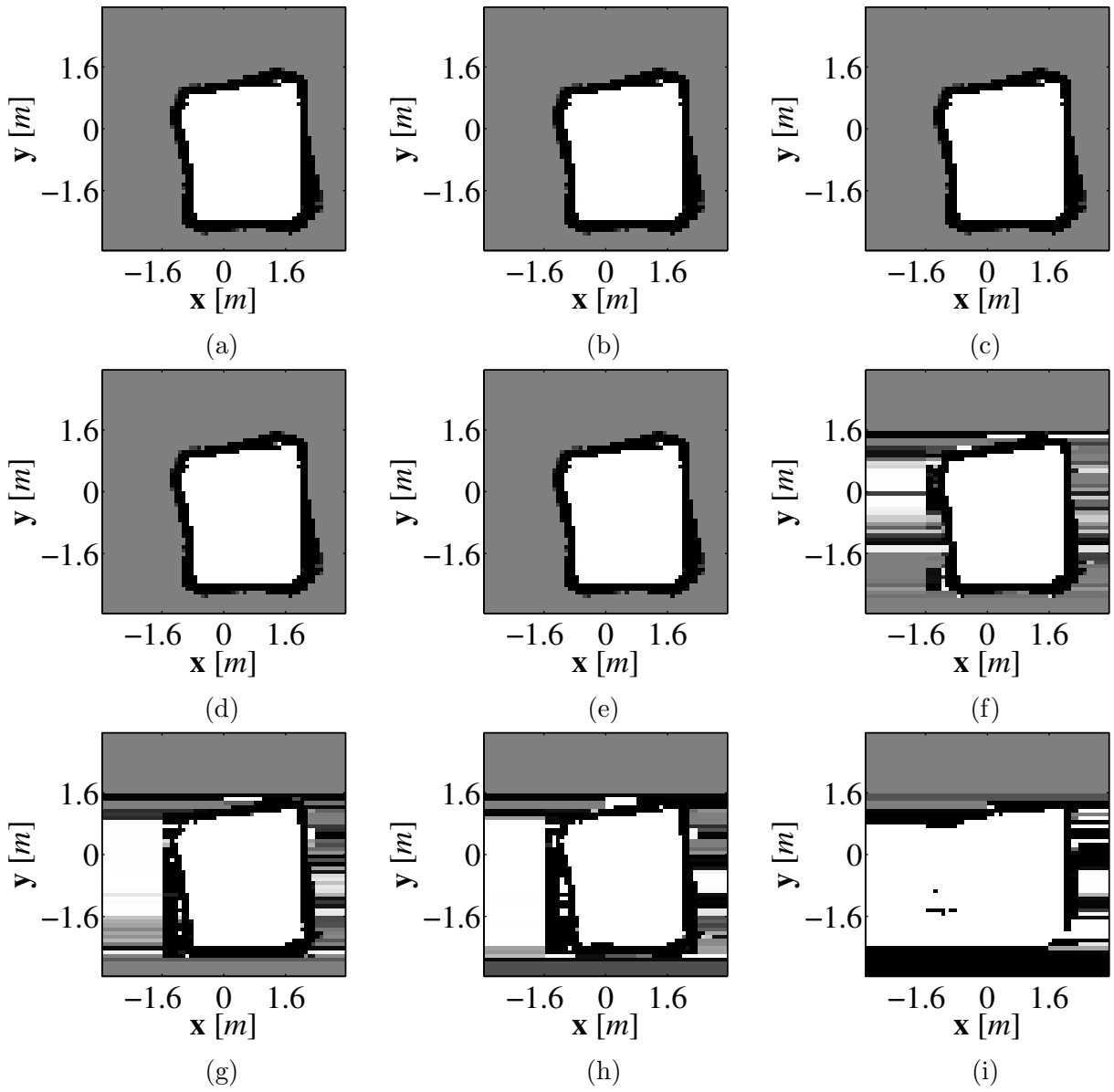


Figure 9.16: A series of final occupancy grids generated by the FastSLAM COG algorithm using the l_2 reconstruction approach and LiDAR data using 75% (a), 70% (b), 60% (c), 50% (d), 40% (e), 30% (f), 25% (g), 20% (h), and 10% (i) of the full amount of data.

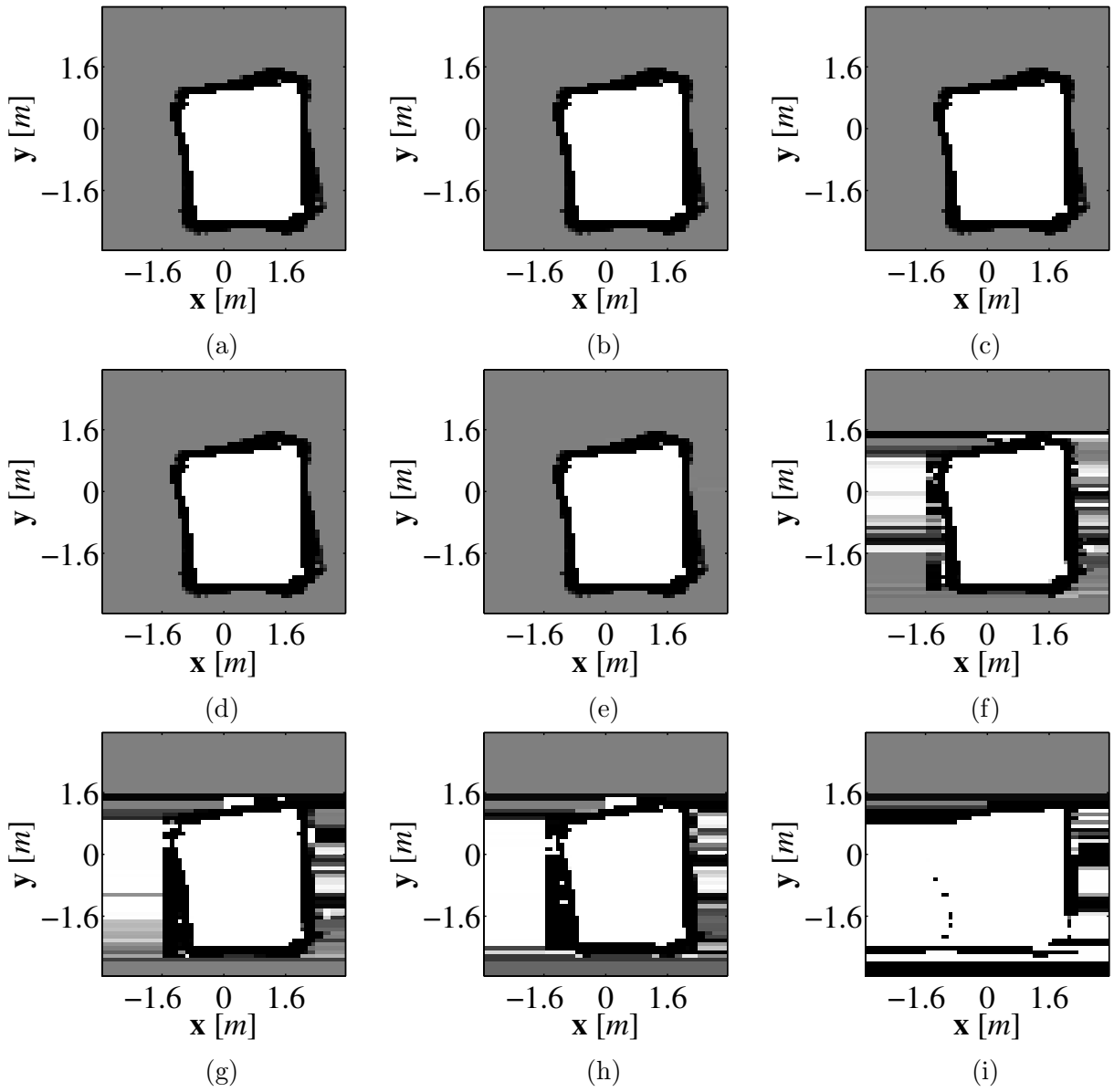


Figure 9.17: A series of final occupancy grids generated by the FastSLAM COG algorithm using the compressed sensing reconstruction approach and LiDAR data using 75% (a), 70% (b), 60% (c), 50% (d), 40% (e), 30% (f), 25% (g), 20% (h), and 10% (i) of the full amount of data.

sensing approach is significantly longer at each compression level than the algorithm using the l_2 reconstruction approach.

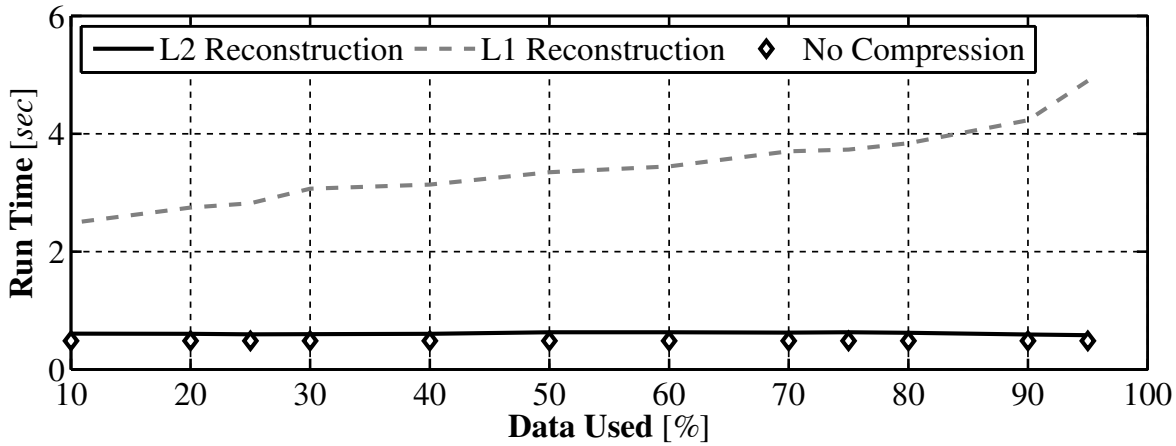
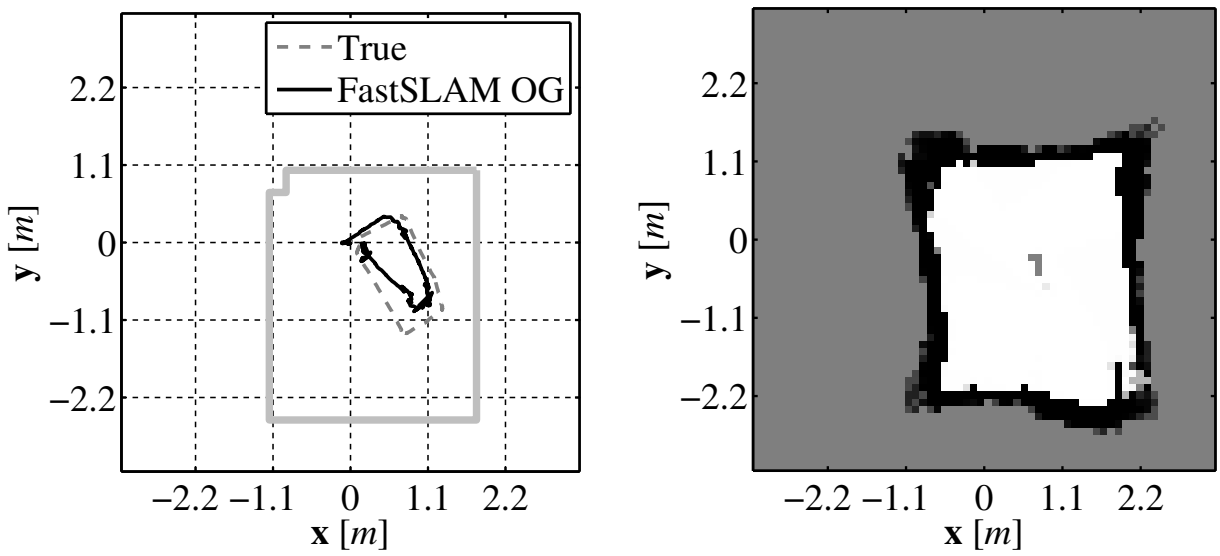


Figure 9.18: The average time to complete a single iteration of the FastSLAM COG algorithm as a function of data used generated with LiDAR data.

9.5.2. Results Using Laser Based Rangefinder

As in the case with the LiDAR results a baseline was generated using the FastSLAM OG approach described in Chapter 7. The baseline vehicle path, along with the true vehicle path, can be seen in Figure 9.19a and the occupancy grid, with a $0.1m$ cell size is shown in Figure 9.19b.

The vehicle path estimate generated by FastSLAM COG algorithm using l_2 reconstruction can be seen in Figure 9.20a and the reconstructed occupancy grid is shown in Figure 9.20b. The generated path estimate using compressed sensing reconstruction approach is shown in Figure 9.21a and the reconstructed occupancy grid is shown in Figure 9.21b. For each of the algorithms 40% of the amount of data used to represent the full occupancy grid is used. Figure 9.22 shows the mean square error (MSE) in the path estimate and reconstructed occupancy grid between the uncompressed form of the algorithm and the compressed form of the algorithm. As Figure 9.22 shows using more than 40% of the data does not decrease

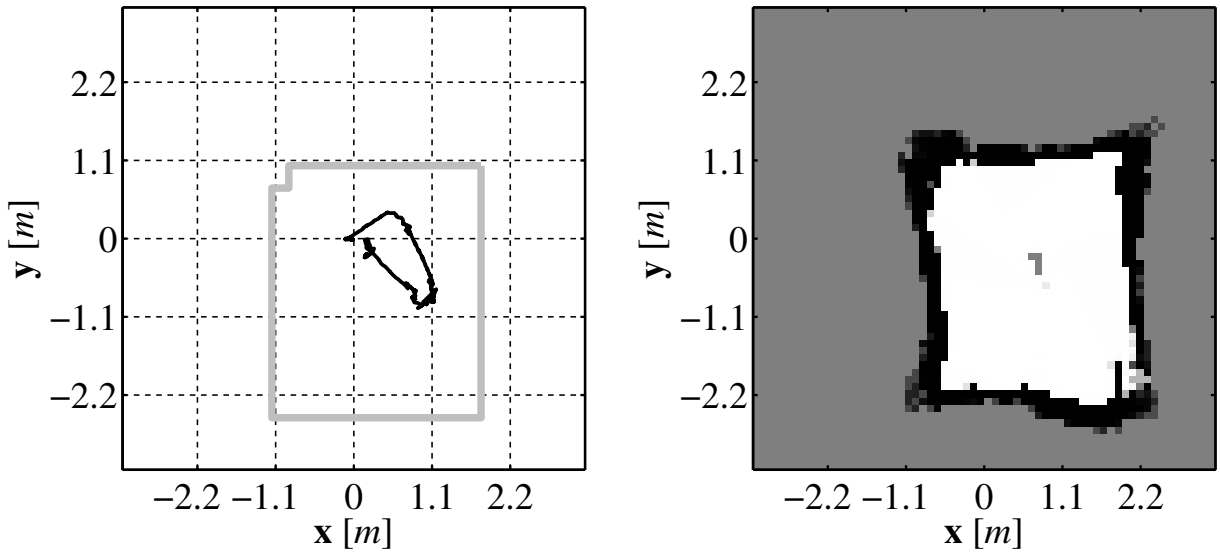


(a) (b)

Figure 9.19: The FastSLAM OG produced path estimate (a) and map (b) used as a baseline generated with custom laser rangefinder.

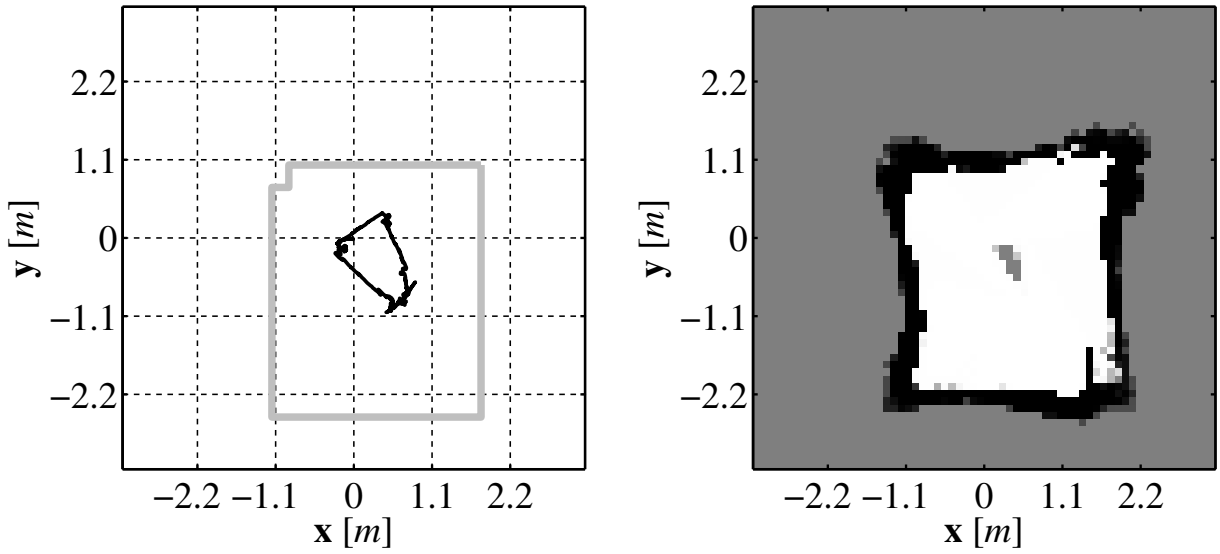
the MSE any more than the error that exists when using 40% of the data.

Figure 9.23 shows a comparison of the average time it takes to run a single iteration of the FastSLAM COG algorithm using the l_2 approach and the compressed sensing approach for varying amounts of data. As seen in these results the average run time of the FastSLAM COG algorithm using the compressed sensing approach is significantly longer at each compression level than the algorithm using the l_2 reconstruction approach. Also, of importance in this result is that by using our laser based rangefinder, which has fewer measurements than the LiDAR sensor, the run time of each of the approach drops. From this result it can be seen that using our custom designed laser based rangefinder we can run the FastSLAM COG algorithm at xHz which for our slow moving underwater vehicle is fast enough to consider running at real-time.



(a) (b)

Figure 9.20: The FastSLAM COG produced path estimate (a) and map (b) generated using the l_2 reconstruction approach with 40% of the data and custom laser rangefinder.



(a) (b)

Figure 9.21: The FastSLAM COG produced path estimate (a) and map (b) generated using the compressed sensing reconstruction approach with 40% of the data and custom laser rangefinder.

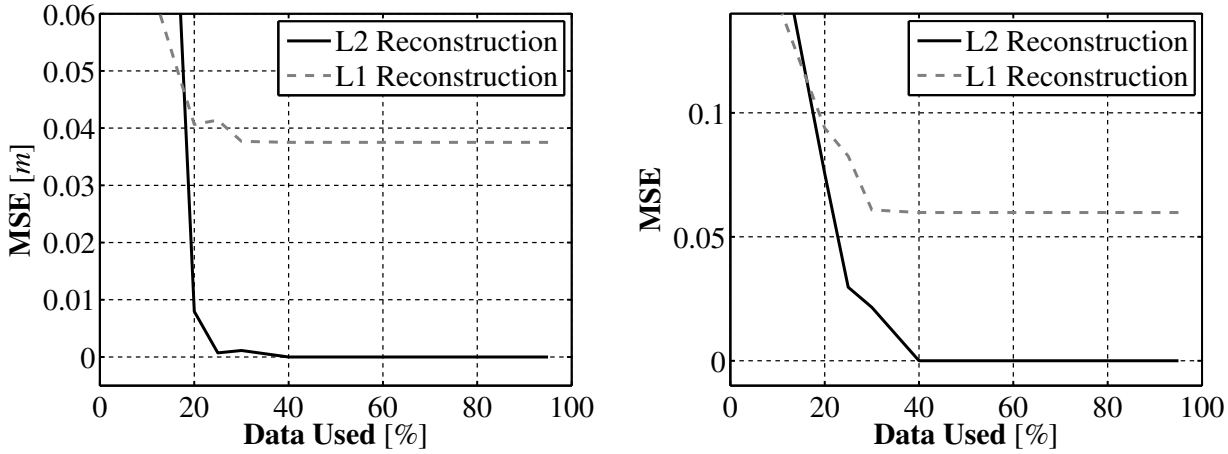


Figure 9.22: The MSE as a function of data percentage for the estimated path (a) and map (b) generated with custom laser rangefinder.

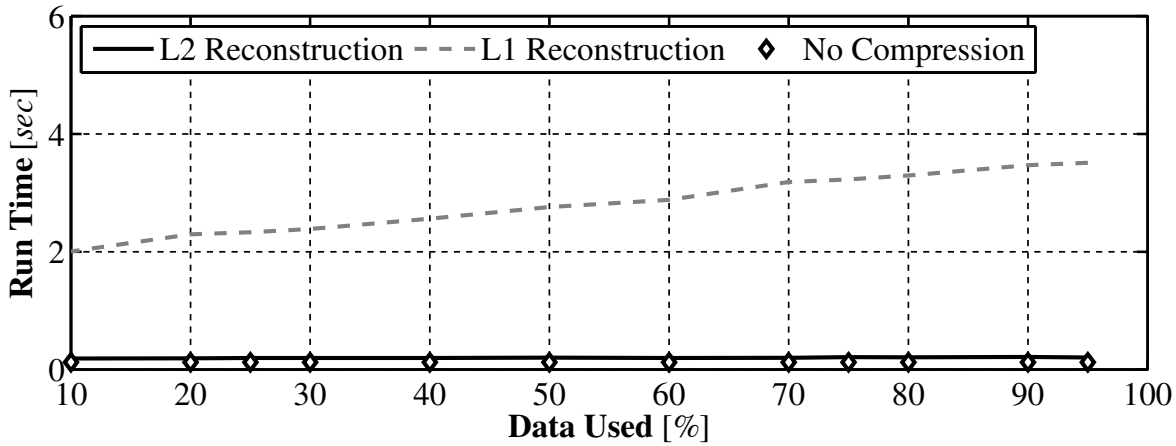


Figure 9.23: The average time to complete a single iteration of the FastSLAM COG algorithm as a function of data used generated with custom laser rangefinder data.

9.5.3. How much compression can be achieved?

There are two key components that affect how much an occupancy grid can be compressed, first is the complexity of the environment in which the UMV is operating. If the environment is simple, such as a rectangular room, then a relatively small amount of data is needed to represent it with respect to a more complex environment. The second component that affects how much an occupancy grid can be compressed is the cell size used. If a large cell size is selected then there are fewer coefficients to represent the environment compared to when a smaller cell size is selected. It follows that when a larger cell size is selected a larger percentage of the coefficients must be used to store the information about the environment, even for simple environments so the grid cannot be compressed as much. To illustrate these two components, three separate environments of increasing complexity where generated and a compressed occupancy grid was generated for each environment using four separate cell sizes ($1.0m$, $0.5m$, $0.2m$, and $0.1m$) and varying amounts of compression. The first environment is a simple rectangular room and an occupancy grid for this environment generated using the full set of data with the smallest cell size is shown in Figure 9.24. The MSE as function of data stored for this environment is shown in Figure 9.25 for each of the four cell size. As expected the environment with the largest cell size, where there are fewer coefficients to store all of the environment information, has the largest error when a small amount of data is used to represent the environment. It can also can be seen that for each of the four cell sizes, as more information is stored, the error in the reconstructed form of the occupancy grid decreases.

A second environment with more complexity is shown in Figure 9.26 which is a rectangular room with complex corners. The MSE between the true occupancy grid and the reconstructed grid as a function of the amount of data stored is shown in Figure 9.27 for each of the cell sizes. The overall behavior of the error is the same as in the simple case with

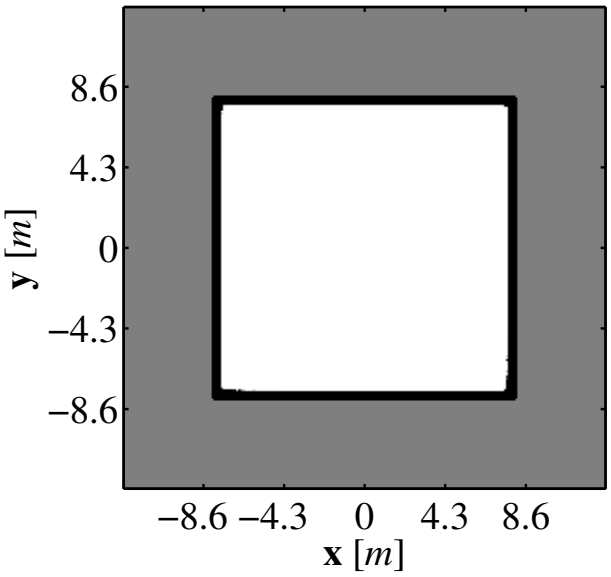


Figure 9.24: Occupancy grid of simple rectangular environment using 100% of the data and a cell size of $0.1m$.

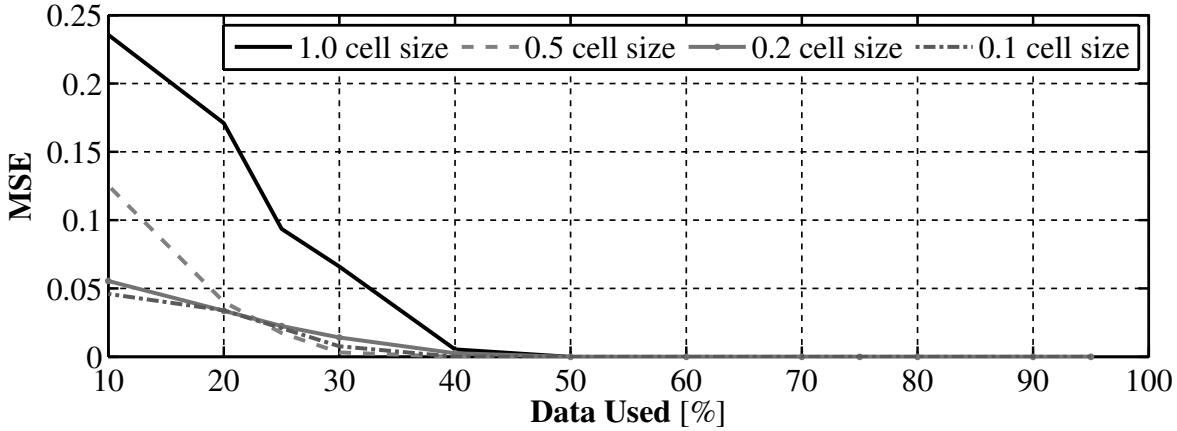


Figure 9.25: MSE between the true occupancy grid of the simple environment and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes.

the largest cell size having the largest error when small amounts of data are used. However, because of the added complexity in this environment, the errors at the lower data percentages are larger than the simple case which is what we expected because more information is required to store the more complex environment.

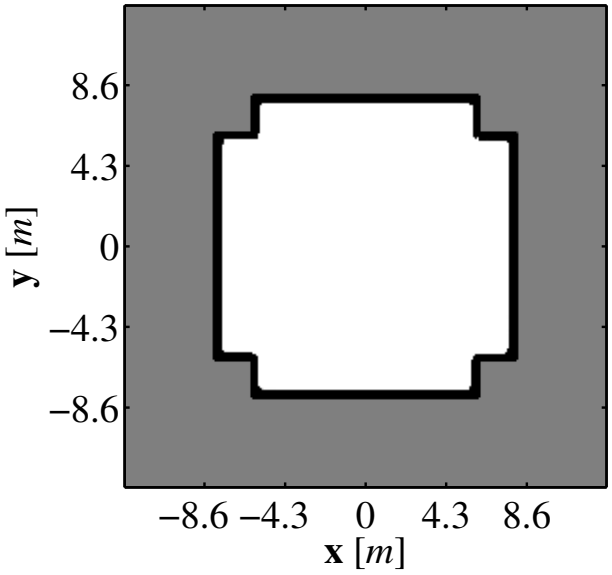


Figure 9.26: Occupancy grid of rectangular environment with complex corners using 100% of the data and a cell size of $0.1m$.

Finally, a complex environment of a rectangular room with complex corners along with rectangular objects located throughout the rectangular room is shown in Figure 9.28. As with the previous two environments the MSE between the true occupancy grid and the reconstructed occupancy as a function of data is shown in Figure 9.29 for each cell size. As expected the use of larger cell sizes causes more errors in the reconstructed form of the grid when small amounts of data are used. Also, the additional complexity causes the errors at lower data percents to be larger than in the more simple environments.

A second view of this data can be seen by examining Figure 9.30 which compares the MSE between the true occupancy grid and the uncompressed form for each of the environments for the 1.0m cell size. From this result it can be seen that each of the environments have

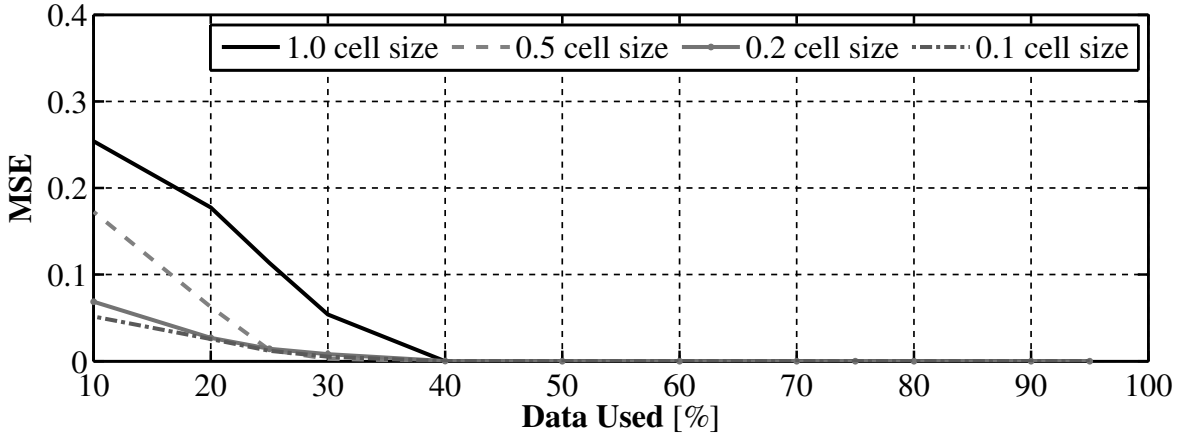


Figure 9.27: MSE between the true occupancy grid of the environment with medium complexity and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes.

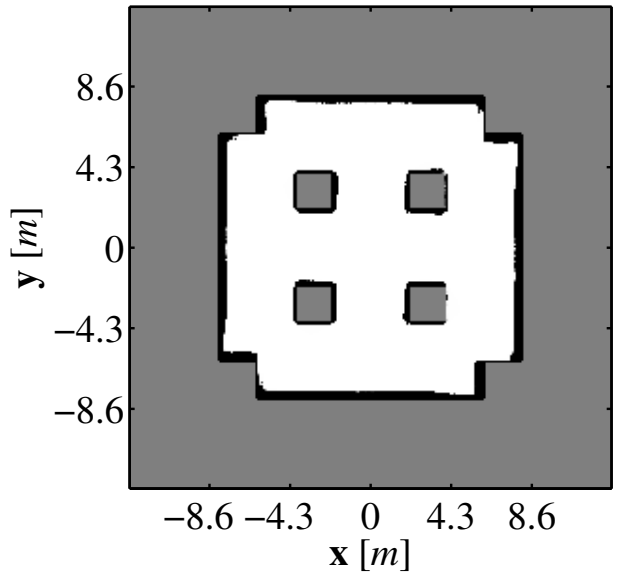


Figure 9.28: Occupancy grid of the complex environment using 100% of the data and a cell size of $0.1m$.

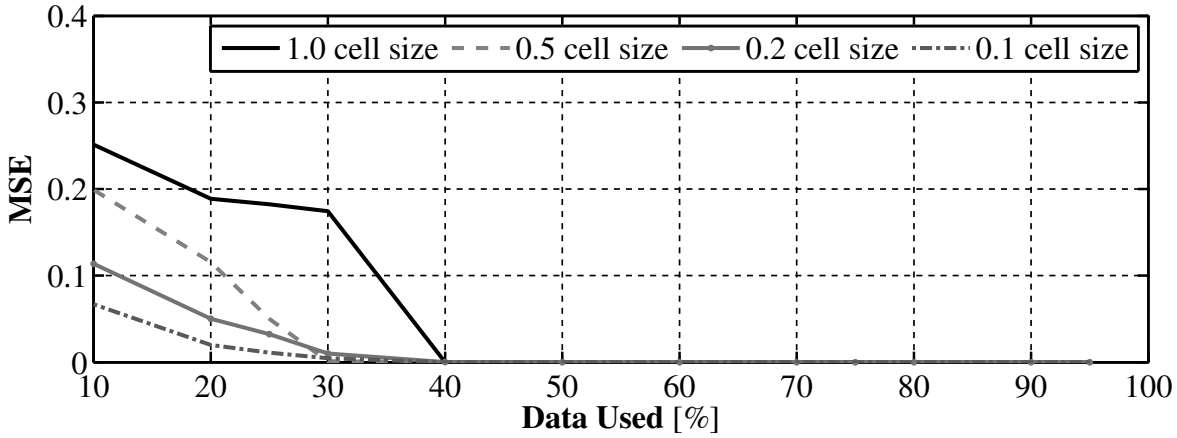


Figure 9.29: MSE between the true occupancy grid of the complex environment and the reconstructed form of the grid as a function of percent of data stored for each of the four cell sizes.

large errors when a small amount of data is used to represent the environment which is what is expected. Also, the error in the complex environment stays the largest as the size of the compressed occupancy grid increases which is also what we would expect.

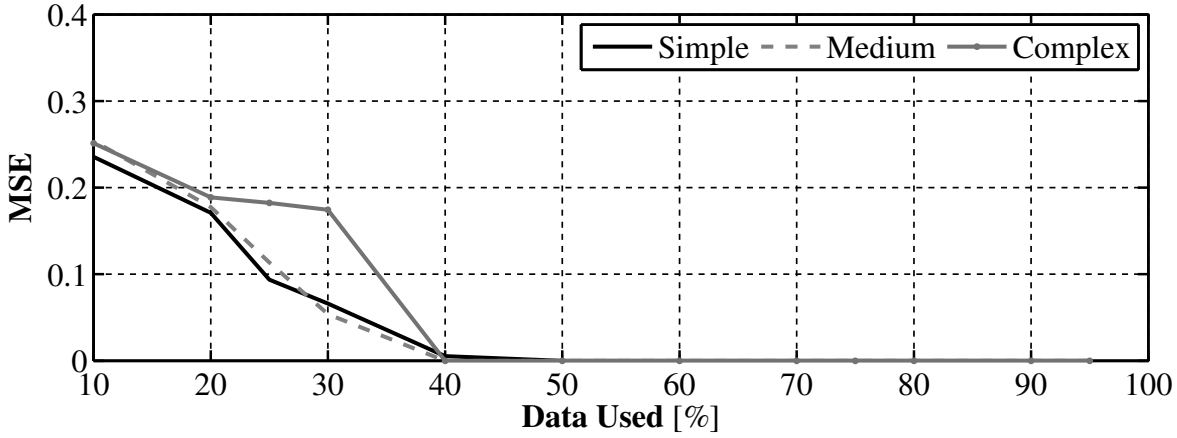


Figure 9.30: MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 1.0m.

The same data can be seen in Figure 9.31 for the cell size of 0.5m. The behavior for this case is the same as for the larger cell size, however the magnitude of the error when smaller amounts of data are used to represent the grid are smaller than in the 1.0m cell size

which we would expect since there are more coefficients to represent the grid, even when the percentage of data is small. The same results are presented in Figure 9.32 and Figure 9.33 for the cell sizes of 0.2m and 0.1m and the overall behavior is what we would expect. In general the error in the complex map is largest when a small amount of data is used to represent the environment and the magnitude in the errors decreases as the cell size decreases.

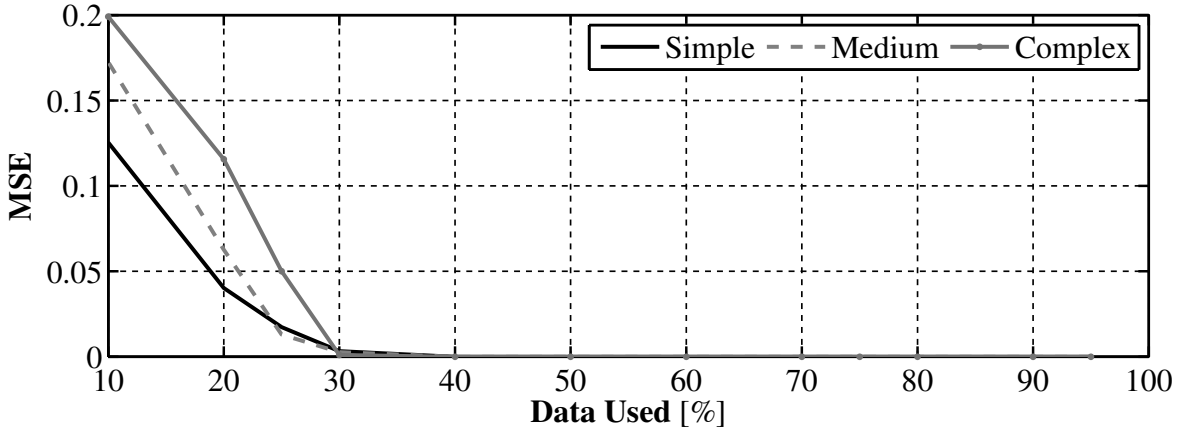


Figure 9.31: MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.5m.

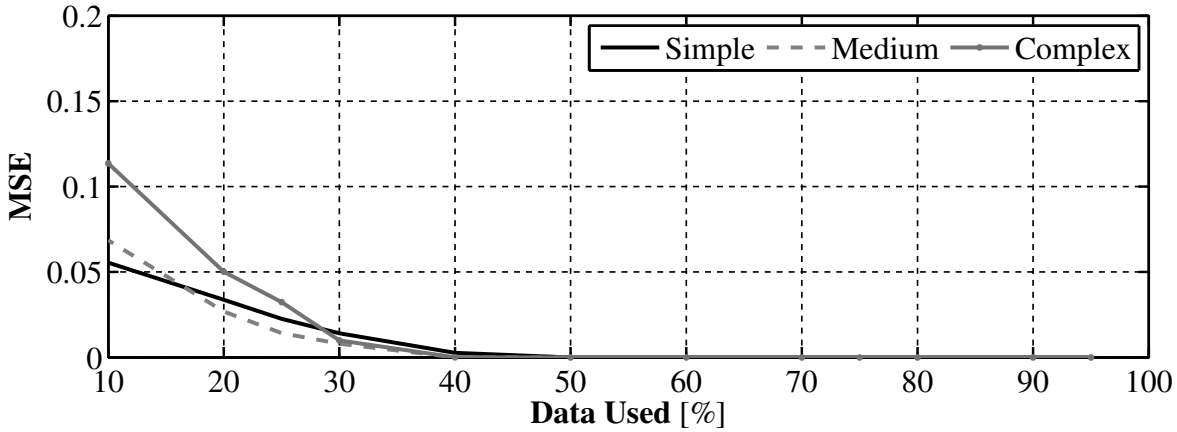


Figure 9.32: MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.2m.

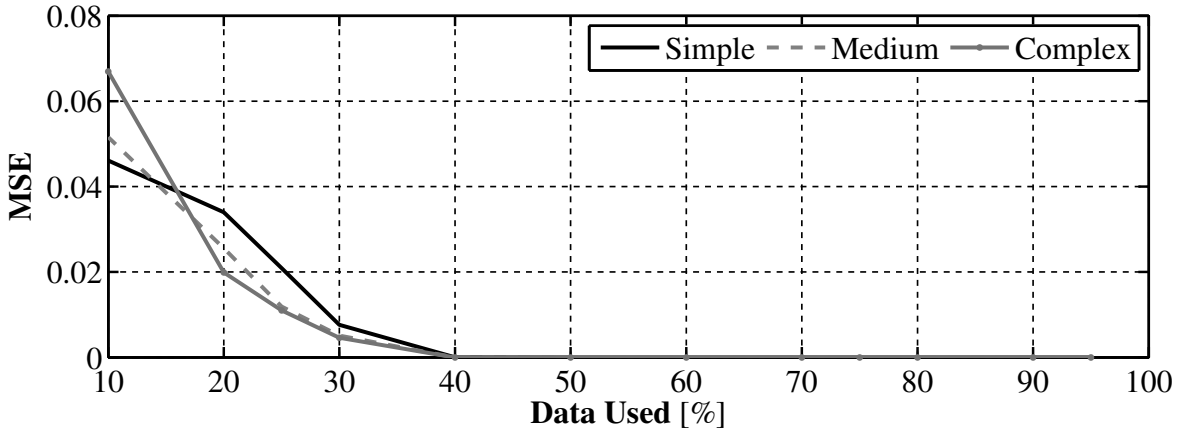


Figure 9.33: MSE between the true occupancy grid and the uncompressed form as a function of compressed size for the cell size of 0.1m.

9.6. Conclusions

In this chapter an approach for solving the SLAM problem was presented that makes use of compressed occupancy grids. The approach is based on an extension to the FastSLAM algorithm that represents an unknown environment by using occupancy grids. The FastSLAM COG algorithm, was presented in Section 9.2 which makes use of a generic compression and uncompression method to solve the SLAM problem. Four specific uncompression methods were examined in Section 9.3 in order to see how well each of the approaches could reconstruct a compressed signal along with an analysis of the time needed for each algorithm to complete. Based on the reconstruction time, two of the four algorithms were used to replace the generic reconstruction method in the FastSLAM COG algorithm and their performance was examined in Section 9.5 using experimental data. From the presented experimental results it was concluded that the FastSLAM COG algorithm using the l_2 reconstruction approach could solve the SLAM problem with almost no errors compared to the FastSLAM OG algorithm while storing only 40% of the data required to store the complete occupancy grid. Further investigation into the effect of using more complex compression and reconstruction

approaches that could lead to higher data compression rates without inducing errors into the SLAM solution.

Chapter 10

Conclusions and Future Research

In this dissertation a real time SLAM solution was developed that can be performed by a low cost UUV equipped with low power and memory restricted computing resources. In Chapter 3 a custom designed underwater rangefinder was developed. The rangefinder is constructed of two laser line generators and a camera. It was shown in the experimental results that the rangefinder can measure the distance to objects with an error whose magnitude is approximately 10% of the actual distance to the object. In Chapter 4 a visual odometry algorithm was presented that makes use of a downward facing camera. The rangefinder and downward facing camera, along with a digital compass, compose the low cost sensor suite that will be used by the UUV to solve the SLAM problem. The selected sensor suite was validated in Chapter 5 and Chapter 6 using two common feature based SLAM algorithms, the EKF SLAM algorithm and the FastSLAM algorithm that makes use of a Rao-Blackwellized particle filter. From the results presented in these two chapters it was shown that the selected sensor suite provides the accurate enough sensor data to enable our low cost UUV to solve the SLAM problem. The SLAM algorithm that serves as the basis for our final implementation was presented in Chapter 7. The SLAM algorithm presented in this chapter, FastSLAM OG, is an extension of the feature based FastSLAM algorithm used to validate our sensor suite. The FastSLAM OG algorithm replaces the feature based map with an occupancy grid

that provides a more complete and detailed picture of the environment surround our UUV. In Chapter 8 an extension to the FastSLAM OG algorithm was presented that changes how the occupancy grid is stored by each particle in the particle filter. The FastSLAM AOG algorithm stores the occupancy grid maintained by each particle in the Haar wavelet basis. Finally, in Chapter 9 an extension of the FastSLAM algorithm that uses compressed occupancy grids as the map was presented. By storing the occupancy grid in compressed form the memory required to perform the algorithm can be significantly reduced. In particular it was shown that the same result can be achieved by the FastSLAM COG algorithm using only 40% of the computer memory required to produce the same result using the FastSLAM OG algorithm.

Future Research

The FastSLAM COG algorithm presented in Chapter 9 made use of a simple alternate basis, the Haar wavelet basis, to represent the vector form of the occupancy grid maintained by each particle and a simple compression scheme that involved selecting the m coefficients with largest magnitude. The FastSLAM COG algorithm could be improved my investigating the effect of using different alternate representations for the vector form of the occupancy grid. By investigating additional alternate representations it could be found that in some other basis the occupancy grid is even more sparse than it is when represented in the Haar wavelet basis. In addition two methods of reconstructing a sparse signal were investigated for use in the FastSLAM COG algorithm, the approach that reconstructs a sparse signal while minimizing the error according to the l_2 norm and the compressed sensing reconstruction approach. The FastSLAM COG algorithm could be improved by investigating other methods of reconstruction sparse signals. In particular the approach presented in [50] which improves on the standard compressed sensing approach by taking into account the fact that in many

situation the non-sparse coefficients of a sparse signal poses a particular structure and by making use of this structure the reconstruction can be improved. A second method, that takes a similar approach, is presented by the authors of [51] who take advantage of the fact that in many situations when basis in which a signal is sparse is multiscale the resulting non-zero coefficients are organized into trees. Using this structure the authors improve the reconstruction over the case when only the knowledge that there are non-zero coefficients is used.

A second way in which the FastSLAM COG algorithm can be improved is to change how the occupancy grid is treated in the algorithm. In Chapter 8 and Chapter 9 the two dimensional occupancy grid is treated as a one dimensional signal. This choice was made for a valid reason in that the compression and reconstruction are significantly simplified by treated the occupancy grid in such a manner. However, there is a drawback, in particular in the calculation of the importance weight for each particle a temporary copy of the occupancy grid in vector form has to be constructed. This is because when treated as a one dimensional signal the only method available of calculating the probabilistic measurement model is by using the raw probability of occupancy values maintained by the vector form of the occupancy grid. However, if the change in basis is performed in two dimensional, for example by using the two dimensional Haar wavelet transform, then the measurement probability can be calculated without access to the raw probability of occupancy values. In particular there are correlation based measurement models that can be used to calculate the importance weight. An example of this approach is referred to as map matching ([101]) in which a sub occupancy is generated using the currently available sensor data. This sub occupancy grid is then cross correlated to find the measurement probability used in the importance weight calculation. In this scenario the cross correlation can be performed in the alternate basis used to represent the occupancy grid in two dimensions.

Bibliography

[1] S. G. Warren, "Optical constants of ice from the ultraviolet to the microwave," *Applied optics*, vol. 23, no. 8, pp. 1206–1225, 1984.

[2] E. J. Stollnitz, A. D. DeRose, and D. H. Salesin, "Wavelets for computer graphics: a primer. 1," *Computer Graphics and Applications, IEEE*, vol. 15, no. 3, pp. 76–84, 1995.

[3] iRobot, "Roomba product page." `http://www.irobot.com/us/learn/home/roomba.aspx` as of March 2013.

[4] K. Muljowidodo, M. A. Rasyid, N. SaptoAdi, and A. Budiyono, "Vision based distance measurement system using single laser pointer design for underwater vehicle," *Indian Journal of Marine science*, vol. 38, no. 3, pp. 324–331, 2009.

[5] G. C. Karras, D. J. Panagou, and K. J. Kyriakopoulos, "Target-referenced localization of an underwater vehicle using a laser-based vision system," in *OCEANS 2006*, pp. 1–6, IEEE, 2006.

[6] A. Bodenmann, B. Thornton, T. Ura, M. Sangekar, T. Nakatani, and T. Sakamaki, "Pixel based mapping using a sheet laser and camera for generation of coloured 3d seafloor reconstructions," in *OCEANS 2010*, pp. 1–5, IEEE, 2010.

[7] M. N. Sangekar, B. Thornton, T. Nakatani, and T. Ura, "Development of a landing algorithm for autonomous underwater vehicles using laser profiling," in *OCEANS 2010 IEEE-Sydney*, pp. 1–7, IEEE, 2010.

[8] M. Kurisu, Y. Yokokohji, Y. Shiokawa, and T. Samejima, "Development of a laser range finder for 3d map-building in rubble," in *Intelligent Robots and Systems, 2004 IEEE International Conference*, vol. 6, pp. 443–448, IEEE, 2004.

[9] M. Kurisu, Y. Yokokohji, and Y. Oosato, "Development of a laser range finder for 3d map-building in rubble the 2nd report: development of the 2nd prototype," in *IEEE international conference on mechatronics and automation*, pp. 1842–1847, 2005.

[10] M. Kurisu, H. Muroi, Y. Yokokohji, and H. Kuwahara, "Development of a laser range finder for 3d map-building in rubble; installation in a rescue robot," in *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pp. 2054–2059, IEEE, 2007.

[11] M. Killpack, T. Deyle, C. Anderson, and C. C. Kemp, "Visual odometry and control for an omnidirectional mobile robot with a downward-facing camera," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 139–146, IEEE, 2010.

[12] L. Piyathilaka and R. Munasinghe, "An experimental study on using visual odometry for short-run self localization of field robot," in *Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on*, pp. 150–155, IEEE, 2010.

[13] L. Piyathilaka and R. Munasinghe, "Multi-camera visual odometry for skid steered field robot," in *Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on*, pp. 189–194, IEEE, 2010.

[14] N. Nourani-Vatani and P. V. K. Borges, "Correlation-based visual odometry for ground vehicles," *Journal of Field Robotics*, vol. 28, no. 5, pp. 742–768, 2011.

[15] M. Dille, B. Grocholsky, and S. Singh, "Outdoor downward-facing optical flow odometry with commodity sensors," in *Field and Service Robotics*, pp. 183–193, Springer, 2010.

[16] S. Basu, "Efficient multiscale template matching with orthogonal wavelet decompositions," tech. rep., Tech. Rep., MIT Media Lab, 1997.

[17] H.-J. Cho and T.-H. Park, "Wavelet transform based image template matching for automatic component inspection," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 9-12, pp. 1033–1039, 2010.

[18] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1, pp. 185–203, 1981.

[19] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision.," in *IJCAI*, vol. 81, pp. 674–679, 1981.

[20] R. J. Adrian, "Twenty years of particle image velocimetry," *Experiments in fluids*, vol. 39, no. 2, pp. 159–169, 2005.

[21] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[22] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*, pp. 167–193, Springer, 1990.

[23] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *Automatic Control, IEEE Transactions on*, vol. 45, no. 3, pp. 477–482, 2000.

[24] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[25] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng, *Simultaneous mapping and localization with sparse extended information filters: Theory and initial results*. Springer, 2002.

[26] T. Lemaire, S. Lacroix, and J. Sola, "A practical 3d bearing-only slam algorithm," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 2449–2454, IEEE, 2005.

[27] J. Weingarten and R. Siegwart, "Ekf-based 3d slam for structured environment reconstruction," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3834–3839, IEEE, 2005.

[28] J. A. Castellanos, J. Martnez, J. Neira, and J. D. Tardos, "Experiments in multisensor mobile robot localization and map building," *Proc. 3rd IFAC Sym. Intell. Auton. Vehicles*, pp. 173–178, 1998.

[29] J.-H. Kim and S. Sukkarieh, "Airborne simultaneous localisation and map building," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1, pp. 406–411, IEEE, 2003.

[30] J. Kim and S. Sukkarieh, "Real-time implementation of airborne inertial-slam," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 62–71, 2007.

[31] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 321–328, IEEE, 2000.

[32] K. P. Murphy, "Bayesian map learning in dynamic environments.," in *NIPS*, pp. 1015–1021, 1999.

[33] M. Montemerlo, *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[34] D. Roller, M. Montemerlo, S. Thrun, and B. Wegbreit, "Fastslam 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[35] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, pp. 593–598, 2002.

[36] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 176–183, Morgan Kaufmann Publishers Inc., 2000.

[37] A. Eliazar and R. Parr, "Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *IJCAI*, vol. 3, pp. 1135–1142, 2003.

[38] J. Choi, S. Ahn, and W. K. Chung, "Robust sonar feature detection for the slam of mobile robot," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3415–3420, IEEE, 2005.

[39] J. Shen, J. Liu, R. Zhao, and X. Lin, "A kd-tree-based outlier detection method for airborne lidar point clouds," in *Image and Data Fusion (ISIDF), 2011 International Symposium on*, pp. 1–4, IEEE, 2011.

[40] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 1929–1934, IEEE, 2005.

[41] J. Xavier, M. Pacheco, D. Castro, A. Ruano, and U. Nunes, "Fast line, arc/circle and leg detection from laser scan data in a player driver," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 3930–3935, IEEE, 2005.

[42] "The player project website." `http://playerstage.sourceforge.net/` as of September 2012.

[43] Y. Li and E. B. Olson, "Structure tensors for general purpose lidar feature extraction," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1869–1874, IEEE, 2011.

[44] M. E. West and V. L. Syrmos, "Navigation of an autonomous underwater vehicle (auv) using robust slam," in *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pp. 1801–1806, IEEE, 2006.

[45] C. M. Clark, C. S. Olstad, K. Buhagiar, and T. Gambin, "Archaeology via underwater robots: Mapping and localization within maltese cistern systems," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pp. 662–667, IEEE, 2008.

[46] J. Salvi, Y. Petillot, S. Thomas, and J. Aulinas, "Visual slam for underwater vehicles using video velocity log and natural landmarks," in *OCEANS 2008*, pp. 1–6, IEEE, 2008.

[47] R. Eustice, H. Singh, J. J. Leonard, M. Walter, and R. Ballard, "Visually navigating the rms titanic with slam information filters.," in *Robotics: Science and Systems*, pp. 57–64, 2005.

[48] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *Information Theory, IEEE Transactions on*, vol. 52, no. 2, pp. 489–509, 2006.

[49] R. Chartrand, "Exact reconstruction of sparse signals via nonconvex minimization," *Signal Processing Letters, IEEE*, vol. 14, no. 10, pp. 707–710, 2007.

[50] V. Cevher, P. Indyk, L. Carin, and R. G. Baraniuk, "A tutorial on sparse signal acquisition and recovery with graphical models."

[51] C. La and M. N. Do, "Signal reconstruction using sparse tree representations," in *Optics & Photonics 2005*, pp. 59140W–59140W, International Society for Optics and Photonics, 2005.

[52] W. Wang, M. J. Wainwright, and K. Ramchandran, "Information-theoretic limits on sparse signal recovery: Dense versus sparse measurement matrices," *Information Theory, IEEE Transactions on*, vol. 56, no. 6, pp. 2967–2979, 2010.

[53] S. Thrun, W. Burgard, D. Fox, *et al.*, *Probabilistic robotics*, vol. 1. MIT press Cambridge, 2005.

[54] M. Montemerlo and S. Thrun, *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics (Springer Tracts in Advanced Robotics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[55] Sony, "Sony fcb-ex11d product page." `http://http://pro.sony.com/bbsc/ssr/cat-industrialcameras/cat-block/product-FCBEX11D/` as of September 2012, 2012. Retrieved September, 2012, from `http://http://pro.sony.com/bbsc/ssr/cat-industrialcameras/cat-block/product-FCBEX11D/`.

[56] Apinex, "Apinex gm-cw02l product page." `http://www.apinex.com/ret2/gmcw02L.html` as of September 2012.

[57] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media Inc., 2008.

[58] C. B. DUANE, "Close-range camera calibration," *Photogrammetric engineering*, vol. 37, no. 8, pp. 855–866, 1971.

[59] J.-Y. Bouguet, "Camera calibration toolbox for matlab." `http://www.vision.caltech.edu/bouguetj/calib_doc/` as of September 2012.

[60] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, pp. 666–673, Ieee, 1999.

[61] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 1106–1112, IEEE, 1997.

[62] W. Garage, "Opencv wiki." `http://opencv.willowgarage.com/wiki/` as of September 2012.

[63] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pp. 281–297, California, USA, 1967.

[64] Hokuyo, "Hokuyo utm-30lx product page." `http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html` as of September 2012.

[65] N. B. Priyantha, *The cricket indoor location system*. PhD thesis, Massachusetts Institute of Technology, 2005.

[66] Hagisonic, *User's Guide: Localization system Stargazer™ for Intelligent Robots*, 2010.

[67] H. C. Woithe, D. Boehm, and U. Kremer, "Improving slocum glider dead reckoning using a doppler velocity log," in *OCEANS 2011*, pp. 1–5, IEEE, 2011.

[68] "Explorer doppler velocty log."

[69] P. Bartlett, D. Wettergreen, and W. R. L. Whittaker, "Design of the scarab rover for mobility and drilling in the lunar cold traps," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, February 2008.

[70] D. Wettergreen, D. Jonak, D. Kohanbash, S. J. Moreland, S. Spiker, J. Teza, and W. Whittaker, "Design and experimentation of a rover concept for lunar crater resource survey," in *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, 2009.

[71] C. M. Gifford, "Low-cost mobile robot localization using only a downward-facing webcam," tech. rep., Technical Report, University of Kansas, USA, 2009.

[72] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[73] A. H. Jazwinski, *Stochastic processes and filtering theory*. Courier Dover Publications, 2007.

[74] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[75] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, jun 1981.

[76] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," in *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, pp. 107–113, IET, 1993.

[77] A. F. Smith and A. E. Gelfand, "Bayesian statistics without tears: a sampling–resampling perspective," *The American Statistician*, vol. 46, no. 2, pp. 84–88, 1992.

[78] A. Elfes, *Occupancy grids: A probabilistic framework for robot perception and navigation*. PhD thesis, Carnegie Mellon University, 1989.

[79] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI magazine*, vol. 9, no. 2, p. 61, 1988.

[80] T. Balch, "Grid-based navigation for mobile robots," *The Robotics Practitioner*, vol. 2, no. 1, pp. 6–11, 1996.

[81] 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.

[82] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910.

[83] O. Diegel, A. Badve, G. Bright, J. Potgieter, and S. Tlale, "Improved mecanum wheel design for omni-directional robots," in *Proc. 2002 Australasian Conference on Robotics and Automation, Auckland*, pp. 117–121, 2002.

[84] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2432–2437, IEEE, 2005.

[85] R. Baraniuk, M. A. Davenport, M. F. Duarte, and C. Hegde, "An introduction to compressive sensing," *Connexions e-textbook*, 2011.

[86] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *Computers, IEEE Transactions on*, vol. C-23, pp. 90–93, Jan 1974.

[87] E. H. Moore, "On the reciprocal of the general algebraic matrix," *Bulletin of the American Mathematical Society*, vol. 26, pp. 394–395, 1920.

[88] R. Penrose, "A generalized inverse for matrices," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, no. 03, pp. 406–413, 1955.

[89] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[90] S. Becker, J. Bobin, and E. J. Candès, "Nesta: a fast and accurate first-order method for sparse recovery," *SIAM Journal on Imaging Sciences*, vol. 4, no. 1, pp. 1–39, 2011.

[91] E. Candes and J. Romberg, "l_1-magic: Recovery of sparse signals via convex programming," 2005.

[92] A. Carmi, P. Gurfil, and D. Kanevsky, "Methods for sparse signal recovery using kalman filtering with embedded pseudo-measurement norms and quasi-norms," *Signal Processing, IEEE Transactions on*, vol. 58, no. 4, pp. 2405–2409, 2010.

[93] S. J. Julier and J. J. LaViola, "On kalman filtering with nonlinear equality constraints," *Signal Processing, IEEE Transactions on*, vol. 55, no. 6, pp. 2774–2784, 2007.

[94] R. G. Brown and P. Y. C. Hwang, *Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 3rd ed.* New York, NY: Wiley, 1997.

[95] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd ed., 2001.

[96] G. Arfken, *Mathematical Methods for Physicists*. Orlando: ap, third ed., 1985.

[97] M. Lee and M. Kaveh, "Fast hadamard transform based on a simple matrix factorization," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34, pp. 1666–1667, Dec 1986.

[98] R. Coifman, F. Geshwind, and Y. Meyer, "Noiselets," *Applied and Computational Harmonic Analysis*, vol. 10, no. 1, pp. 27–44, 2001.

[99] Mathworks, "Moore-penrose pseudoinverse of matrix." `http://www.mathworks.com/help/matlab/ref/pinv.html` as of December 2013.

[100] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50. Siam, 1997.

[101] B. Schiele and J. L. Crowley, "A comparison of position estimation techniques using occupancy grids," *Robotics and autonomous systems*, vol. 12, no. 3, pp. 163–171, 1994.

[102] S. B. Williams, *Efficient solutions to autonomous mapping and navigation problems.* PhD thesis, The University of Sydney, 2001.

Appendix A

EKF SLAM Details

A.1. Introduction

As stated in Chapter 5 the EKF SLAM algorithm represents the estimates the distribution that represents the full SLAM state as a Gaussian distribution. This distribution can be represented by a mean vector and covariance matrix. The mean vector of the full SLAM state contains a pose estimate of the UMV given as

$$\boldsymbol{\xi}_k = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^{\mathrm{T}} \tag{A.1}$$

where $(x_k, y_k) \in \mathbb{R}^2$ is the two dimensional Cartesian coordinates of the UMV in some frame of reference and $\theta_k \in [-\pi, \pi)$ is the heading of the UMV with respect to the positive horizontal axis of the frame of reference. The mean vector also contains an estimate of the map of the environment in which the UMV operates. The EKF SLAM algorithm represents the environment in which the UMV operates using a set of landmarks, thus the map is given as

$$\mathbf{M} = \begin{Bmatrix} \mathbf{m}_1, & \mathbf{m}_2, & \ldots, & \mathbf{m}_n \end{Bmatrix}, \tag{A.2}$$

where $\mathbf{m}_i = (x_i, y_i) \in \mathbb{R}^2$, $i = 1, \ldots, n$ is the two dimensional Cartesian coordinates that make up the position of the ith landmark and $n \in \mathbb{N}$ is the total number of landmarks in $\mathbf{M}$ that have been observed before time step k. Combining these two components the joint

mean vector maintained by the EKF SLAM algorithm is given as

$$\mathbf{x}_k \triangleq \begin{bmatrix} \boldsymbol{\xi}_k & \mathbf{M} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{\xi}_k & \mathbf{m}_1 & \mathbf{m}_2 & \cdots & \mathbf{m}_n \end{bmatrix}^{\mathrm{T}} \tag{A.3}$$

where $\mathbf{x}_k \in \mathbb{R}^{3+2n}$. The complete EKF SLAM algorithm is presented in Chapter 5 however it is repeated in Algorithm A.1 for quick reference.

A.2. Model Verification

As stated in Chapter 5 the EKF SLAM algorithm uses an EKF ([73]) to estimate the distribution that represents the full SLAM state. In order to make use of the EKF it is assumed that the state transition model and measurement model are of the from given by

$$\mathbf{x}_k = \mathbf{g}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \boldsymbol{\epsilon}_k, \tag{A.4}$$

$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \boldsymbol{\delta}_k, \tag{A.5}$$

where $\mathbf{g} : \mathbb{R}^{(3+2n)\times(3+2n)} \times \mathbb{R}^c \rightarrow \mathbb{R}^{3+2n}$ is the state transition model that describes how the system evolves between time steps based on the previous estimate $\mathbf{x}_{k-1}$ and the current control input $\mathbf{u}_k$. In order to use the EKF the additive noise in the state transition model $\boldsymbol{\epsilon}_k$ is assumed to be Gaussian with a covariance $\mathbf{R}_k$. The measurement model $\mathbf{h} : \mathbb{R}^{3+2n} \rightarrow \mathbb{R}^m$ describes the location of the extracted features that correspond to landmarks in the environment based on the current state estimate and the additive noise $\boldsymbol{\delta}_k$ is assumed to be Gaussian with a covariance $\mathbf{Q}_k$. In this remainder of this section we will show that the state transition model and measurement model that we used in our implementation meet the form required to make use of the EKF.

A.2.1. State Transition Model

The state transition model of the system describes how the entire SLAM state evolves between time steps. The state transition model of our system is based on two components:

Algorithm A.1 Overview of the feature based EKF SLAM algorithm repeated for reference.

procedure EKFSLAM($\hat{\mathbf{x}}_{k-1}, \boldsymbol{\Sigma}_{k-1}, \mathbf{u}_k, \mathbf{z}_k$)
    Data:
        $\hat{\mathbf{x}}_{k-1}$ the mean vector from the previous time step
        $\boldsymbol{\Sigma}_{k-1}$ the covariance matrix from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of observations

    Result:
        $\hat{\mathbf{x}}_k$ the current mean vector
        $\boldsymbol{\Sigma}_k$ the current covariance matrix

    $\bar{\hat{\mathbf{x}}}_k \leftarrow \mathbf{g}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k\right)$
    $\bar{\boldsymbol{\Sigma}}_k \leftarrow \mathbf{G}_k \boldsymbol{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}} + \mathbf{R}_k$
    $\mathbf{c}_k \leftarrow$ DataAssociate($\bar{\hat{\mathbf{x}}}_k, \mathbf{z}_k$)
    for $i = 1, \ldots, m$ **do**
        if $\mathbf{c}_k(i) \neq -1$ **then**
            $\mathbf{K}_k \leftarrow \bar{\boldsymbol{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k \bar{\boldsymbol{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k\right)^{-1}$
            $\hat{\mathbf{x}}_k \leftarrow \bar{\hat{\mathbf{x}}}_k + \mathbf{K}_k \left(\mathbf{z}_k(i) - \mathbf{h}\left(\bar{\hat{\mathbf{x}}}_k\right)\right)$
            $\boldsymbol{\Sigma}_k \leftarrow (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\boldsymbol{\Sigma}}_k$
            $\bar{\hat{\mathbf{x}}}_k \leftarrow \hat{\mathbf{x}}_k$
            $\bar{\boldsymbol{\Sigma}}_k \leftarrow \boldsymbol{\Sigma}_k$
        end if
    end for
    for $i = 1, \ldots, m$ **do**
        if $\mathbf{c}_k(i) = -1$ **then**
            $\hat{\mathbf{x}}_k^+ \leftarrow \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{f}\left(\hat{\mathbf{x}}_k, \mathbf{z}_k(i)\right) \end{bmatrix}$
            $\boldsymbol{\Sigma}_k^+ = \begin{bmatrix} \boldsymbol{\Sigma}_k & \mathbf{A}_k^{\mathrm{T}} \\ \mathbf{A}_k & \mathbf{B}_k \end{bmatrix}$
        end if
    end for
end procedure

i) the state transition model of the UMV based on our use of a downward facing camera to provide visual odometry measurements and compass as described in Chapter 4 and ii) the assumption that the environment that the UMV operates in is static thus the location of landmarks do not change over time. If we assume that the visual odometry measurements and compass provide perfect measurements then the state transition model of the UMV is given as

$$\begin{bmatrix} x_k \\ y_k, \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \delta x_k \cos\theta_{k-1} - \delta y_k \sin\theta_{k-1} \\ y_{k-1} + \delta x_k \sin\theta_{k-1} + \delta y_k \cos\theta_{k-1} \\ \theta_{k,m} \end{bmatrix}, \tag{A.6}$$

where the control vector for the UMV state transition model contains the translations measured by the visual odometry system and the heading of the vehicle measured by the compass, $\mathbf{u}_k = \begin{bmatrix} \delta x_k & \delta y_k & \theta_{k,m} \end{bmatrix}^{\mathrm{T}}$. In reality each of the control measurements is corrupted by noise, thus we define the noisy control vector as

$$\tilde{\mathbf{u}}_k = \begin{bmatrix} \delta\tilde{x}_k \\ \delta\tilde{y}_k \\ \tilde{\theta}_{k,m} \end{bmatrix} = \begin{bmatrix} \delta x_k + \epsilon_{k,\delta x} \\ \delta y_k + \epsilon_{k,\delta y} \\ \theta_{k,m} + \epsilon_{k,\theta} \end{bmatrix}, \tag{A.7}$$

where $\epsilon_{k,\delta x} \in \mathbb{R}$, $\epsilon_{k,\delta y} \in \mathbb{R}$, and $\epsilon_{k,\theta} \in \mathbb{R}$ is additive noise corrupting the true measurement value. Since we are making use of the EKF to estimate the distribution we assume the noise on each of the control signals is Gaussian, thus the noise corrupting the each of the control signals is given as

$$\epsilon_{k,\delta x} \sim \mathcal{N}\left(0, \sigma^2_{k,\delta x}\right), \quad \epsilon_{k,\delta y} \sim \mathcal{N}\left(0, \sigma^2_{k,\delta y}\right), \quad \text{and} \quad \epsilon_{k,\theta} \sim \mathcal{N}\left(0, \sigma^2_{k,\theta}\right), \tag{A.8}$$

where $\sigma_{k,\delta x} \in \mathbb{R}$, $\sigma_{k,\delta y} \in \mathbb{R}$, and $\sigma_{k,\theta} \in \mathbb{R}$ is the standard deviation of the noise corrupting the controls. By replacing the ideal control inputs in (A.6) with the noisy controls the state transtion model of the UMV becomes

$$\begin{bmatrix} x_k \\ y_k, \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \delta\tilde{x}_k \cos\theta_{k-1} - \delta\tilde{y}_k \sin\theta_{k-1} \\ y_{k-1} + \delta\tilde{x}_k \sin\theta_{k-1} + \delta\tilde{y}_k \cos\theta_{k-1} \\ \tilde{\theta}_{k,m} \end{bmatrix}, \tag{A.9}$$

which can be expanded using (A.7) to become

$$\begin{bmatrix} x_k \\ y_k, \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + (\delta x_k + \epsilon_{k,\delta x})\cos\theta_{k-1} - (\delta y_k + \epsilon_{k,\delta y})\sin\theta_{k-1} \\ y_{k-1} + (\delta x_k + \epsilon_{k,\delta x})\sin\theta_{k-1} + (\delta y_k + \epsilon_{k,\delta y})\cos\theta_{k-1} \\ \theta_{k,m} + \epsilon_{k,\theta} \end{bmatrix}. \tag{A.10}$$

If we rearrange (A.10) and include the state transition model of the map based on the assumption that the environment is static, then the state transition model of the full state is given as

$$\underbrace{\begin{bmatrix} x_k \\ y_k, \\ \theta_k \\ x_{k,1} \\ y_{k,1} \\ \vdots \\ x_{k,n} \\ y_{k,n} \end{bmatrix}}_{\mathbf{x}_k} = \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ 0 \\ x_{k-1,1} \\ y_{k-1,1} \\ \vdots \\ x_{k-1,n} \\ y_{k-1,n} \end{bmatrix} + \begin{bmatrix} \delta x_k \cos\theta_{k-1} - \delta y_k \sin\theta_{k-1} \\ \delta x_k \sin\theta_{k-1} + \delta y_k \cos\theta_{k-1} \\ \theta_{k,m} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{g}(\mathbf{x}_{k-1},\mathbf{u}_k)} + \underbrace{\begin{bmatrix} \epsilon_{k,\delta x}\cos\theta_{k-1} - \epsilon_{k,\delta y}\sin\theta_{k-1} \\ \epsilon_{k,\delta x}\sin\theta_{k-1} + \epsilon_{k,\delta y}\cos\theta_{k-1} \\ \epsilon_{k,\theta} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\boldsymbol{\epsilon}_k \sim \mathcal{N}(0,\mathbf{R}_k)}. \tag{A.11}$$

As given in Chapter 5 Section 5.3.1 the covariance matrix of the state transition model $\mathbf{R}_k$ is defined as

$$\mathbf{R}_k \triangleq \mathbf{V}_k \mathbf{M}_k \mathbf{V}_k^{\mathrm{T}}, \tag{A.12}$$

where $\mathbf{M}_k \in \mathbb{R}^{c\times c}$ is the covariance matrix of the control input and $\mathbf{V}_k \in \mathbb{R}^{(3+2n)\times c}$ is the Jacobian of $\mathbf{g}(\cdot)$ with respect to the control input. For the visual odoemetry system the covariance matrix is given as

$$\mathbf{M}_k = \begin{bmatrix} \sigma_{k,\delta x}^2 & 0 & 0 \\ 0 & \sigma_{k,\delta y}^2 & 0 \\ 0 & 0 & \sigma_{k,\theta}^2 \end{bmatrix}, \tag{A.13}$$

and the Jacobian of $\mathbf{g}(\cdot)$ with respect to the control input is given as

$$\mathbf{V}_k = \begin{bmatrix} \cos\theta_{k-1} & -\sin\theta_{k-1} & 0 \\ \sin\theta_{k-1} & \cos\theta_{k-1} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}. \tag{A.14}$$

A.2.2. Measurement Model

The second key model of the system that must be verified is the measurement model. The measurement defines how a sensor measurement is related to the SLAM state that is being estimated by the EKF. The measurement vector $\mathbf{z}_k$ contains m observations that are features in the raw sensor data that correspond to m landmarks in the environment. In our implementation each observation in $\mathbf{z}_k$ is composed of a range and bearing to a landmark in the environment, thus a single observation is given as

$$\mathbf{z}_k(i) = \begin{bmatrix} r_i \\ \phi_i \end{bmatrix} \tag{A.15}$$

where $i = 1, \ldots, m$, r is the range to the feature and ϕ is the bearing to the feature. The observation in the vehicles local frame of reference can be seen in Figure A.1a and the relationship between the feature location in the observation and the landmark that it corresponds to in the global frame of reference can be seen in Figure A.1b. We must now define how the range and bearing to the feature in the observation relate to the location of a landmark in the global frame of reference. Each observation in $\mathbf{z}_k$ corresponds to a landmark in the environment that is a part of the map maintained by the EKF. This relationship is stored in the vector of correspondence values $\mathbf{c}_k$. If the ith observation in $\mathbf{z}_k$ corresponds to the jth landmark in the map then $\mathbf{c}_k(i) = j$. For the case where $\mathbf{c}_k(i) = j$ if the sensor measurement is perfect then the ith observation is defined as

$$\mathbf{z}_k(i) = \begin{bmatrix} r_i \\ \phi_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_{k,j} - x_k)^2 + (y_{k,j} - y_k)^2} \\ \theta_k - \arctan 2\left(y_{k,j} - y_k, x_{k,j} - x_k\right) \end{bmatrix}, \tag{A.16}$$

where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. This gives the range of the observation as the Euclidean distance between the vehicle location and the location of the landmark that corresponds to the observation and the bearing is the relative angle to the landmarks location. Again, as in the state of the control inputs, it is assumed that the observation vector is corrupted by

noise. The measurement with noise added is given as

$$\mathbf{z}_k(i) = \begin{bmatrix} r_i \\ \phi_i \end{bmatrix} + \begin{bmatrix} \delta_{k,r} \\ \delta_{k,\phi} \end{bmatrix}, \tag{A.17}$$

where $\delta_{k,r} \in \mathbb{R}$ and $\delta_{k,\phi}$ are additive noise that are corrupting the true observation values. Based on the use of the EKF we assume that the noise corrupting the observation is Gaussian thus the additive noise corrupting the observation is given as

$$\delta_{k,r} \sim \mathcal{N}\left(0, \sigma_{k,r}^2\right), \quad \text{and} \quad \delta_{k,\phi} \sim \mathcal{N}\left(0, \sigma_{k,\phi}^2\right), \tag{A.18}$$

where $\sigma_{k,r} \in \mathbb{R}$ is the standard deviation of the noise corrupting the range measurement and $\sigma_{k,\phi}$ is the standard deviation of the noise corrupting the bearing measurement. Finally, the measurement model of the system is given as

$$\underbrace{\begin{bmatrix} r_i \\ \phi_i \end{bmatrix}}_{\mathbf{z}_k(i)} = \underbrace{\begin{bmatrix} \sqrt{(x_{k,j} - x_k)^2 + (y_{k,j} - y_k)^2} \\ \theta_k - \arctan 2\left(y_{k,j} - y_k, x_{k,j} - x_k\right) \end{bmatrix}}_{\mathbf{h}(\mathbf{x}_k)} + \underbrace{\begin{bmatrix} \delta_{k,r} \\ \delta_{k,\phi} \end{bmatrix}}_{\boldsymbol{\delta}_k \sim \mathcal{N}(0, \mathbf{Q}_k)}, \tag{A.19}$$

which matches the required form and the covariance matrix of the measurement model is given as

$$\mathbf{Q}_k = \begin{bmatrix} \sigma_{k,r}^2 & 0 \\ 0 & \sigma_{k,\phi}^2 \end{bmatrix}. \tag{A.20}$$

Now that our state transition model and measurement model have been shown to match the form required by the EKF in the following section the details of the prediction phase of EKF SLAM will be presented.

A.3. Prediction

The first phase of the EKF SLAM algorithm is referred to as the prediction phase. During the prediction phase the mean vector and covariance matrix of the estimate are updated by incorporating the current control input $\mathbf{u}_k$ into the estimate. The control input is incorporated through the state transition model of the system. The first component that is updated is the mean vector and the mean vector prediction will now be presented in detail.

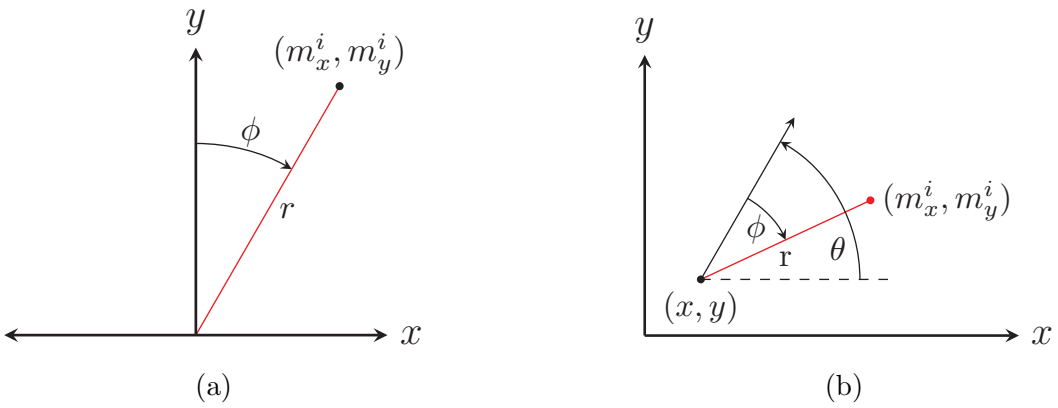


Figure A.1: Measurement model of the system in the vehicle's local frame (a) and in the worlds global frame (b).

A.3.1. Mean Vector Prediction

The first step of the EKF SLAM solution is referred to as mean prediction and is based on the state transition model of the system. During this step a "predicted" value of the state is calculated. The predicted value of the state is recursively generated based on estimate from the previous time step $\mathbf{x}_{k-1}$ and the current control input $\mathbf{u}_k$. The predicted value of the mean is generated according to the state transition model of the system, thus the predicted value of the estimate is given as

$$\bar{\hat{\mathbf{x}}}_k = \mathbf{g}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k\right). \tag{A.21}$$

Using the previously defined state transition model of the system (A.6), the state predicted value of the mean vector is generated according to

$$\bar{\hat{\mathbf{x}}}_k = \begin{bmatrix} \bar{\hat{x}}_k \\ \bar{\hat{y}}_k, \\ \bar{\hat{\theta}}_k \\ \bar{\hat{x}}_{k,1} \\ \bar{\hat{y}}_{k,1} \\ \vdots \\ \bar{\hat{x}}_{k,n} \\ \bar{\hat{y}}_{k,n} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1} \\ \hat{y}_{k-1} \\ 0 \\ \hat{x}_{k-1,1} \\ \hat{y}_{k-1,1} \\ \vdots \\ \hat{x}_{k-1,n} \\ \hat{y}_{k-1,n} \end{bmatrix} + \begin{bmatrix} \delta x_k \cos\theta_{k-1} - \delta y_k \sin\theta_{k-1} \\ \delta x_k \sin\theta_{k-1} + \delta y_k \cos\theta_{k-1} \\ \theta_{k,m} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{A.22}$$

The predicted mean vector of the estimate is based the state transition model of the system which make use of the noisy control inputs. The noisy control inputs add uncertainty to the estimate, thus the covariance matrix must be updated to included this additional uncertainty. The construction of the predicted value of the covariance matrix is presented in the following section.

A.3.2. Covariance Matrix Prediction

According to the standard EKF equations the predicted value of the covariance matrix is generated according to

$$\bar{\boldsymbol{\Sigma}}_k = \mathbf{G}_k \boldsymbol{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}} + \mathbf{R}_k. \tag{A.23}$$

where $\boldsymbol{\Sigma}_{k-1}$ is the covariance matrix from the previous time step, $\mathbf{R}_k$ is the covariance matrix of the state transition model as defined in Chapter 5 Section 5.3.1, and $\mathbf{G}_k$ is the Jacobian of the state transition model of the with respect to the state. Using this definition and our previously presented state transition model $\mathbf{G}_k$ is given as

$$\begin{aligned}
\mathbf{G}_k &= \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k} \\
&= \begin{bmatrix}
1 & 0 & -\delta x_k \sin\hat{\theta}_k - \delta y_k \cos\hat{\theta}_k & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 1 & \delta x_k \cos\hat{\theta}_k - \delta y_k \sin\hat{\theta}_k & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1
\end{bmatrix}.
\end{aligned} \tag{A.24}$$

Using the form of $\mathbf{G}_k$ the covariance matrix prediction takes current uncertainty in the estimate, $\mathbf{G}_k \boldsymbol{\Sigma}_{k-1} \mathbf{G}_k^{\mathrm{T}}$, and adds to that the uncertainty that results from the current noisy

control inputs, $\mathbf{R}_k$. Once the prediction phase has been performed the second phase of the EKF SLAM algorithm uses the current set of observations to *correct* the estimate produced using just the control input. The details of this correction phase are presented in the following section.

A.4. Correction

The second stage of the EKF SLAM solution is referred to as the correction stage. During this phase the current set of observations are used to adjust the mean vector of the estimate while reducing the uncertainty in the estimate. The first component of the estimate that is corrected is the mean vector and the details of the mean vector correction are provided in the following section.

A.4.1. Mean Vector Correction

For each observation in $\mathbf{z}_k$ the mean vector is corrected according to

$$\hat{\mathbf{x}}_k = \bar{\hat{\mathbf{x}}}_k + \mathbf{K}_k \left(\mathbf{z}_k(i) - \mathbf{h}\left(\bar{\hat{\mathbf{x}}}_k \right) \right), \tag{A.25}$$

where $i = 1, \ldots, m$. One of the key components of the correction stage is $\mathbf{K}_k$ in (A.25) which is referred to as the Kalman gain matrix. The Kalman gain matrix is a weighting matrix that determines how highly the observation should be weighted when it is incorporated into the estimate. The level of importance of the observation is based on the covariance matrices of the state transition model and the measurement model. Using these covariance matrices the Kalman gain matrix is given as

$$\mathbf{K}_k = \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^{\mathrm{T}} + \mathbf{Q}_k \right)^{-1}, \tag{A.26}$$

where $\mathbf{H}_k$ is the Jacobian of $\mathbf{h}(\cdot)$ with respect to the system state. Using previously defined measurement model, (A.16), when the ith observation corresponds to the jth landmark $\mathbf{H}_k$

is defined as

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\bar{\hat{\mathbf{x}}}_k} = \begin{bmatrix} \frac{\bar{\hat{x}}_k - \bar{\hat{x}}_{k,j}}{\sqrt{q}} & \frac{\bar{\hat{y}}_k - \bar{\hat{y}}_{k,j}}{\sqrt{q}} & 0 & 0 & 0 & \cdots & \frac{\bar{\hat{x}}_{k,j} - \bar{\hat{x}}_k}{\sqrt{q}} & \frac{\bar{\hat{y}}_{k,j} - \bar{\hat{y}}_k}{\sqrt{q}} & \cdots & 0 & 0 \\ \frac{\bar{\hat{y}}_{k,j} - \bar{\hat{y}}_k}{q} & \frac{\bar{\hat{x}}_k - \bar{\hat{x}}_{k,j}}{q} & 0 & 0 & 0 & \cdots & \frac{\bar{\hat{y}}_k - \bar{\hat{y}}_{k,j}}{q} & \frac{\bar{\hat{x}}_{k,j} - \bar{\hat{x}}_k}{q} & \cdots & 0 & 0 \end{bmatrix}, \tag{A.27}$$

where $q \triangleq \left(\bar{\hat{x}}_k - \bar{\hat{x}}_{k,j}\right)^2 + \left(\bar{\hat{y}}_k - \bar{\hat{y}}_{k,j}\right)^2$. In the final step of the EKF SLAM algorithm the covariance matrix is corrected using the current observation. The details of the covariance matrix correction are presented in the following section.

A.4.2. Covariance Matrix Correction

In the final step of the EKF SLAM algorithm the uncertainty in the estimate is reduced by correcting the covariance matrix of the estimate. As in the mean vector correction the covariance matrix correction makes use of the Kalman gain matrix, which is defined in Section A.4.2. Using the previously defined matrices the covariance matrix correction is performed according to the standard EKF correction equation which is given as

$$\hat{\mathbf{\Sigma}}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\bar{\hat{\mathbf{\Sigma}}}_k. \tag{A.28}$$

With the details of each of the standard EKF estimation equations defined for our system in the following section the details of the state augmentation component of EKF SLAM, a step that is specific to using the EKF in SLAM, are presented.

A.5. State Augmentation

As discussed in Chapter 5 an additional step is present in the EKF SLAM algorithm that doesn't exist in the standard EKF. This step is referred to as "augmentation" and pertains to the addition of new landmarks to the map maintained by the estimate. When a feature is extracted from the sensor data that does not correspond to a landmark currently held in the

map the new landmark must be added to the estimate. The augmentation step makes use of the inverse sensor model which estimates the position of a landmark, in the global frame of reference, based on the current estimate and the observation that corresponds to the newly observed landmark. For our system if the ith observation corresponds to a landmark that is not stored in the map then the inverse sensor model is defined as

$$\hat{\mathbf{m}}_{k,n+1} = \begin{bmatrix} \hat{x}_{k,n+1} \\ \hat{y}_{k,n+1} \end{bmatrix} = \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{z}_k(i)) = \begin{bmatrix} \hat{x}_k + r_i \cos\left(\hat{\theta}_k - \phi_i\right) \\ \hat{y}_k + r_i \sin\left(\hat{\theta}_k - \phi_i\right) \end{bmatrix} \tag{A.29}$$

where the new landmark is located at $\hat{\mathbf{m}}_{k,n+1} = (\hat{x}_{k,n+1}, \hat{y}_{k,n+1}) \in \mathbb{R}^2$ based on the measurement $\mathbf{z}_k(i) = \begin{bmatrix} r_i & \phi_i \end{bmatrix}^{\mathrm{T}}$ which does not correspond to any of the landmarks currently stored in the map. With the invsere sensor model defined in the following section the augmentation of the mean vector is presented.

A.5.1. Mean Vector Augmentation

The first step in the augmentation stage is to augment the mean vector of the estimate. In this step the map maintained by the estimate is updated to include the new landmark. The new landmark is added to the mean vector of the estimate according to

$$\hat{\mathbf{x}}_k^+ = \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{z}_k) \end{bmatrix}, \tag{A.30}$$

where $\hat{\mathbf{x}}_k^+ \mathbb{R}^{5+2n}$ is the mean of the estimate after the augmentation has been performed. With the new landmark added to the map maintained by the estimate, the final step of the EKF SLAM algorithm is to added the uncertainty corresponding to the new landmark to the estimate. This is performed by augmenting the covariance matrix of the estimate and the details are provided in the following section.

A.5.2. Covariance Matrix Augmentation

The augmentation of the estimate's covariance matrix is more complex than the augmentation of the mean. A computationally efficient method of augmenting the covariance matrix is presented in [102]. The covariance matrix of the estimate before the augmentation takes the form

$$\boldsymbol{\Sigma}_k^+ = \begin{bmatrix} \boldsymbol{\Sigma}_k & \mathbf{A}_k \\ \mathbf{A}_k^{\mathrm{T}} & \mathbf{B}_k \end{bmatrix} \tag{A.31}$$

where $\boldsymbol{\Sigma}_k^+ \in \mathbb{R}^{(5+2n)\times(5+2n)}$ is the covariance matrix after the augmentation. The matrix $\mathbf{A}_k \in \mathbb{R}^{2\times(3+2n)}$ is the cross covariance terms between the each of the elements already present in the mean vector and the new landmark location. The matrix $\mathbf{A}_k$ is defined as

$$\mathbf{A}_k \triangleq \mathbf{F}_{k,x}\boldsymbol{\Sigma}_k, \tag{A.32}$$

where $\mathbf{F}_{k,x} \in \mathbb{R}^{2\times 3}$ is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the state and it propagates the uncertainty in the current estimate into the cross covariance terms of the new landmark. Using the previously defined inverse sensor model the matrix $\mathbf{F}_{k,x}$ is given as

$$\begin{aligned} \mathbf{F}_{k,x} &= \left.\frac{\partial f}{\partial x}\right|_{\hat{\mathbf{x}}_k, \mathbf{z}_k(i)} \\ &= \begin{bmatrix} 1 & 0 & -r_i \sin\left(\hat{\theta}_k - \phi_i\right) \\ 0 & 1 & r_i \cos\left(\hat{\theta}_k - \phi_i\right) \end{bmatrix}. \end{aligned} \tag{A.33}$$

The matrix $\mathbf{B}_k$ generates the uncertainty in new landmark location. The uncertainty in the new landmark location is a combination of the uncertainty in the current estimate maintained by the EKF along with the uncertainty related to using a noisy sensor to observe the new landmark. The result is that the matrix $\mathbf{B}_k$ is defined as

$$\mathbf{B}_k \triangleq \mathbf{F}_{k,x}\boldsymbol{\Sigma}_k\mathbf{F}_{k,x}^{\mathrm{T}} + \mathbf{F}_{k,z}\mathbf{Q}_k\mathbf{F}_{k,z}^{\mathrm{T}}, \tag{A.34}$$

where $\mathbf{F}_{k,z} \in \mathbb{R}^{2\times 2}$ is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the observation. Using the previously defined inverse sensor model $\mathbf{F}_{k,z}$ is defined as

$$\mathbf{F}_{k,z} = \left.\frac{\partial f}{\partial z}\right|_{\mathbf{x}_k,\mathbf{z}_k} \tag{A.35}$$

$$= \begin{bmatrix} \cos\left(\hat{\theta}_k - \phi_i\right) & r_i \sin\left(\hat{\theta}_k - \phi_i\right) \\ \sin\left(\hat{\theta}_k - \phi_i\right) & -r_i \cos\left(\hat{\theta}_k - \phi_i\right) \end{bmatrix}. \tag{A.36}$$

Now that all of the details of the EKF SLAM algorithm have been presented, the following chapter contains the details related to the classic feature-based FastSLAM algorithm for our UMV.

Appendix B

FastSLAM Details

B.1. Introduction

As stated in Chapter 6 the FastSLAM ([33]) algorithm uses a Rao-Blackwellized Particle Filter to solve the SLAM problem. The FastSLAM algorithm estimates the joint distribution that represents the trajectory of the UMV and the map of the environment which is given by

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{B.1}$$

where $\mathbf{u}_{1:k}$ is the set of control input and $\mathbf{z}_{1:k}$ is the set of observations. The FastSLAM algorithm then factors the joint distribution, (B.1), as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right) p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right). \tag{B.2}$$

where $p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}, \mathbf{c}_{1:k}\right)$ is estimated using a particle filter where each particles maintains its own copy of the map of the environment. The classic FastSLAM algorithm is a feature based solution to the SLAM problem so the map of the environment is composed of n landmarks that represent areas of the environment that can be easily extracted as features in the raw sensor data. As a result the map maintained by pth particle is defined as

$$\mathbf{M}^{[p]} = \left\{ \begin{array}{cccc} \mathbf{m}_1^{[p]}, & \mathbf{m}_2^{[p]}, & \ldots, & \mathbf{m}_n^{[p]} \end{array} \right\}, \tag{B.3}$$

where $\mathbf{m}_i^{[p]} = (x_i, y_i) \in \mathbb{R}^2$, $i = 1, \ldots, n$ is the two dimensional Cartesian coordinates that make up the position of the ith landmark and $n \in \mathbb{N}$ is the total number of landmarks in $\mathbf{M}^{[p]}$ that have been observed before time step k. The FastSLAM algorithm represents each landmark using a Gaussian distribution that is represented using a mean vector $\hat{\mathbf{m}}_{k,i}^{[p]}$ and a covariance matrix $\mathbf{\Sigma}_{k,i}^{[p]}$. Since each landmark is assumed to be a Gaussian distribution the location of each landmark in the map is estimated using an EKF. As a result the pth particle in the particle filter is defined as

$$\mathcal{X}_k^{[p]} \triangleq \left[\begin{array}{ccccccc} \boldsymbol{\xi}_k^{[p]} & \hat{\mathbf{m}}_{k,1}^{[p]} & \mathbf{\Sigma}_{k,1}^{[p]} & \cdots & \hat{\mathbf{m}}_{k,n}^{[p]} & \mathbf{\Sigma}_{k,n}^{[p]} \end{array} \right], \tag{B.4}$$

where $\hat{\mathbf{m}}_{k,j}^{[p]} \in \mathbb{R}^2$, $j = 1, \ldots, n$ and $\mathbf{\Sigma}_{k,j}^{[p]} \in \mathbb{R}^{2\times 2}$, $j = 1, \ldots, n$ are the mean and covariance of the jth landmark estimate. The FastSLAM algorithm was previously presented in Chapter 6 however, it is displayed again in Algorithm B.1 and Algorithm B.2 for reference. In the first step of the algorithm a new set of particle poses are generated based on the probabilistic motion model of the vehicle and the details of that are presented in the following section.

B.2. Pose Sampling

In the first step of the FastSLAM algorithm a set of potential particles are generated. The particles are generated by sampling from the probabilistic motion model of the UMV, thus the pth particle is generated according to

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right), \tag{B.5}$$

where $\boldsymbol{\xi}_k^{[p]}$ is the pose of the pth particle, $\boldsymbol{\xi}_{k-1}^{[p]}$ is the pose of the from the previous time step on which the new pose is based, and $\mathbf{u}_k$ is the current control input. The probabilistic motion model of the UMV makes uses of the state transition model of the UMV and the probabilistic model of noise present on the control input. For our system the state transition model of the UMV is based on the use of a downward facing camera that provides visual odometry

Algorithm B.1 Overview of the feature based FastSLAM algorithm repeated for reference.

procedure FastSLAMOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
  Data:
    $\mathcal{X}_{k-1}$ the particles from the previous time step
    $\mathbf{u}_k$ the current control input
    $\mathbf{z}_k$ the current set of observations

  Result:
    $\mathcal{X}_k$ the new set of particles generated at the current time step

  $\overline{\mathcal{X}}_k \leftarrow \{\}$
  for $p \leftarrow 1, P$ **do**
    $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
    $\mathbf{c}_k^{[p]} \leftarrow$ DataAssociate($\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k$)
    for $i = 1, \ldots, n$ **do**
      if i is not in $\mathbf{c}_k^{[p]}$ **then**
        $\hat{\mathbf{m}}_{k,i}^{[p]} \leftarrow \hat{\mathbf{m}}_{k-1,i}^{[p]}$
        $\boldsymbol{\Sigma}_{k,i}^{[p]} \leftarrow \boldsymbol{\Sigma}_{k-1,i}^{[p]}$
      end if
    end for
    $w_k^{[p]} = 1$
    for $i = 1, \ldots, m$ **do**
      if $\mathbf{c}_k(i) \neq -1$ **then**
        $\overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \hat{\mathbf{m}}_{k-1,\mathbf{c}_k(i)}^{[p]}$
        $\overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \boldsymbol{\Sigma}_{k,\mathbf{c}_k(i)}^{[p]}$
        $\mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_{k,\mathbf{c}_k(i)}^{\mathrm{T}} \left(\mathbf{H}_{k,\mathbf{c}_k(i)} \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_{k,\mathbf{c}_k(i)}^{\mathrm{T}} + \mathbf{Q}_k\right)^{-1}$
        $\hat{\mathbf{m}}_{k,\mathbf{c}_k(i)}^{[p]} \leftarrow \overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]} + \mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \left(\mathbf{z}_k(\mathbf{c}_k(i)) - \mathbf{h}\left(\overline{\hat{\mathbf{m}}}_{k,\mathbf{c}_k(i)}^{[p]}\right)\right)$
        $\boldsymbol{\Sigma}_{k,\mathbf{c}_k(i)}^{[p]} = \left(\mathbf{I} - \mathbf{K}_{k,\mathbf{c}_k(i)}^{[p]} \mathbf{H}_k\right) \overline{\boldsymbol{\Sigma}}_{k,\mathbf{c}_k(i)}^{[p]}$
        $w_k^{[p]} = w_k^{[p]} \times \eta \frac{1}{\sqrt{\left|2\pi \mathbf{Q}_k^{[p]}\right|}} \exp\left(-\frac{1}{2} \frac{\left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)^{\mathrm{T}} \left(\mathbf{z}_k - \mathbf{h}\left(\boldsymbol{\xi}_k^{[p]}, \hat{\mathbf{m}}_{k-1,j}^{[p]}\right)\right)}{\mathbf{Q}_k^{[p]}}\right)$
      end if
    end for

Algorithm B.2 Overview of the feature based FastSLAM algorithm repeated for reference Part2.

```
        for i = 1, ..., m do
            if c_k(i) = −1 then
                m̂^[p]_{k,n+1} ← f(ξ^[p]_k, z_k(j))
                Σ^[p]_{k,n+1} ← F_{k,z} Q_k F^T_{k,z}
            end if
        end for
        X̄^[p]_k ← [ ξ^[p]_k   M^[p]_k   w^[p] ]
    end for
    X_k ← {}
    for p = 1, n do
        select j with probability proportional to w^[p]
        X^[p]_k ← X̄^[j]_k
    end for
end procedure
```

measurements and a compass that provides a global heading, Chapter 4. Assuming perfect sensor measurements the pose of our UMV evolves between time steps according to

$$\begin{bmatrix} x_k \\ y_k, \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \delta x_k \cos\theta_{k-1} - \delta y_k \sin\theta_{k-1} \\ y_{k-1} + \delta x_k \sin\theta_{k-1} + \delta y_k \cos\theta_{k-1} \\ \theta_{k,m} \end{bmatrix}, \tag{B.6}$$

where the control input is $\mathbf{u}_k = \begin{bmatrix} \delta x_k & \delta y_k & \theta_{k,m} \end{bmatrix}^{\mathrm{T}}$. In reality the control inputs are corrupted by noise, thus the control inputs with additive noise are given as

$$\tilde{\mathbf{u}}_k = \begin{bmatrix} \delta\tilde{x}_k \\ \delta\tilde{y}_k \\ \tilde{\theta}_{k,m} \end{bmatrix} = \begin{bmatrix} \delta x_k + \epsilon_{k,\delta x} \\ \delta y_k + \epsilon_{k,\delta y} \\ \theta_{k,m} + \epsilon_{k,\theta} \end{bmatrix}. \tag{B.7}$$

The first step in generating a new pose for the pth particle is a new control input is generated, $\hat{\mathbf{u}}_k$ by purturbing the control vector that is generated by the visual odometry system and compass. In this step additional noise, that is sampled from the distribution that represents the noise present on the control vector, is added to $\mathbf{u}_k$. In the FastSLAM algorithm, as opposed to the EKF SLAM algorithm where the control noise is assumed to be Gaussian, there is no assumption made on the distribution that describes the noise on the control inputs.

In our implementation we assume that the noise on the control inputs is Gaussian, however the only requirement of the FastSLAM algorithm is that the distribution that describes the control noise must be able to be sampled from. Using our assumption that the noise on the control input is Gaussian, the generation of $\hat{\mathbf{u}}_k$ is given as

$$\hat{\mathbf{u}}_k = \begin{bmatrix} \delta\hat{x}_k \\ \delta\hat{y}_k \\ \hat{\theta}_{k,m} \end{bmatrix} = \begin{bmatrix} \delta x_k + \textbf{sample\_gaussian}\left(\sigma^2_{k,\delta x}\right) \\ \delta y_k + \textbf{sample\_gaussian}\left(\sigma^2_{k,\delta y}\right) \\ \theta_{k,m} + \textbf{sample\_gaussian}\left(\sigma^2_{k,\theta}\right) \end{bmatrix}, \tag{B.8}$$

where **sample_gaussian** (b^2) samples from a zero mean Gaussian distribution with standard deviation of b. The implementation of **sample_gaussian** (b^2) is taken from [53] and shown in Algorithm B.3. The function call to **rand**(x, y) in Algorithm B.3 returns a uniformly distributed value in the range of $[x, y]$.

Algorithm B.3 Overview of the algorithm to sample from a Gaussian distribution.

procedure SAMPLEGAUSS(b^2)
  Data:
    b the standard deviation of the Gaussian noise

  Result:
    y the noise value sampled from the Gaussian distribution

  $x \leftarrow 0$
  $i \leftarrow 0$
  for $i < 12$ **do**
    $x \leftarrow x +$ **rand** $(-b, b)$
    $i \leftarrow i + 1$
  end for

  $y \leftarrow \frac{1}{2}x$
end procedure

The artificially inflated control vector, $\hat{\mathbf{u}}_k$, is then used as the control input for the state transition model of the UMV along with the pose from the previous time step, $\boldsymbol{\xi}^{[p]}_{k-1}$. Using

this approach the new particle pose is generated according to

$$\boldsymbol{\xi}_k^{[p]} = \begin{bmatrix} x_k^{[p]} \\ y_k^{[p]} \\ \theta_k^{[p]} \end{bmatrix} = \begin{bmatrix} x_{k-1}^{[p]} + \delta\hat{x}_k^{[p]} \cos\theta_{k-1}^{[p]} - \delta\hat{y}_k^{[p]} \sin\theta_{k-1}^{[p]} \\ y_{k-1}^{[p]} + \delta\hat{x}_k^{[p]} \sin\theta_{k-1}^{[p]} + \delta\hat{y}_k^{[p]} \cos\theta_{k-1}^{[p]} \\ \hat{\theta}_{k,m}^{[p]} \end{bmatrix}. \tag{B.9}$$

Once a new pose has been generated for each particle and added to $\overline{\mathcal{X}}_k$, in the second step of the FastSLAM algorithm the estimate of each landmark in $\mathbf{M}^{[p]}$ is updated and the details of that step are presented in the following section.

B.3. Landmark Estimate Update

As previously stated each particle maintains a set of n EKF's, one for each landmark whose location is being estimated by the particle. In the second stage of the FastSLAM algorithm each of the landmark estimates are updated based on the set of observations at time step k. Before the landmark estimate are updated the correspondence value of each observation in $\mathbf{z}_k$ is calculated and placed into $\mathbf{c}_k$. The correspondence vector contains the identifier for the landmark that the observation corresponds to, thus if the i observation in $\mathbf{z}_{1:k}$ corresponds to the jth landmark in $\mathbf{M}_k^{[p]}$ then $\mathbf{c}_k(i) = j$. If the jth landmark in $\mathbf{M}_k^{[p]}$ that is not observed at time step k the estimate remains unchanged, so the mean vector and covariance matrix are just updated using their previous values as given by

$$\hat{\mathbf{m}}_{k,j}^{[p]} = \hat{\mathbf{m}}_{k-1,j}^{[p]} \quad \text{and} \quad \boldsymbol{\Sigma}_{k,j}^{[p]} = \boldsymbol{\Sigma}_{k-1,j}^{[p]}. \tag{B.10}$$

If the jth landmark is observed at time step k then the estimate is updated using the standard EKF update equations and the first step is to generate a predicted mean vector. The details of this step are presented in the following section.

B.3.1. Prediction Phase

The mean prediction phase of the EKF is based on the state transition model of the system. In the feature estimate EKF the state transition model is simple and based on the assumption that the landmarks in the environment are static. Using this assumption the predicted value of the mean vector is just the estimate of the mean from the previous time step

$$\bar{\hat{\mathbf{m}}}_{k,j}^{[p]} = \hat{\mathbf{m}}_{k-1,j}^{[p]}. \tag{B.11}$$

As we are assuming that the landmark, whose position is being estimated, is static the covariance matrix prediction is also very simple. The predicted value of the covariance matrix takes the uncertainty in the estimate from the previous time step and adds to that the uncertainty generated by using noisy control inputs. However, since we are assuming that there is no control input, based on the static landmark assumption, the predicted value of the covariance matrix is just the covariance matrix that resulted from the previous time step and is given by

$$\bar{\mathbf{\Sigma}}_{k,j}^{[p]} = \mathbf{\Sigma}_{k-1,j}^{[p]}. \tag{B.12}$$

Now that the predicted form of the mean vector and covariance matrix have been generated the second phase of the EKF uses the feature observation to update the position estimate and uncertainty of the landmark that is observed at time step k. The first step in this phase is the mean correction step and those details are presented in the following section.

B.3.2. Correction

The correction stage of the landmark estimate is based on the measurement model of our system, previous described in Appendix A Section A.2.2. The measurement model relates the current observation to the state of the landmark position estimate. In our case the

observation is a range and bearing to the landmark. Using the geometry of our coordinate system as shown in Appendix A Figure A.1 the measurement model is given as

$$\underbrace{\begin{bmatrix} r_{k,i} \\ \phi_{k,i} \end{bmatrix}}_{\mathbf{z}_k(i)} = \underbrace{\begin{bmatrix} \sqrt{\left(\hat{x}_{k,j}^{[p]} - x_k^{[p]}\right)^2 + \left(\hat{y}_{k,j}^{[p]} - y_k^{[p]}\right)^2} \\ \theta_k^{[p]} - \arctan 2\left(\hat{y}_{k,j}^{[p]} - y_k^{[p]}, \hat{x}_{k,j}^{[p]} - x_k^{[p]}\right) \end{bmatrix}}_{\mathbf{h}\left(\hat{\mathbf{m}}_{k,j}\right)}, \tag{B.13}$$

where the jth landmark to which the ith observation corresponds to is located at $\hat{\mathbf{m}}_{k,j}^{[p]} = \left(\hat{x}_{k,j}^{[p]}, \hat{y}_{k,j}^{[p]}\right)$ and the pose estimate of the particle is $\boldsymbol{\xi}_k^{[p]} = \begin{bmatrix} x_k^{[p]} & y_k^{[p]} & \theta_k^{[p]} \end{bmatrix}^{\mathrm{T}}$.

Using the measurement model the mean vector of the estimate is corrected according to

$$\hat{\mathbf{m}}_{k,j}^{[p]} = \bar{\hat{\mathbf{m}}}_{k,j}^{[p]} + \mathbf{K}_{k,j}^{[p]}\left(\mathbf{z}_k(j) - \mathbf{h}\left(\bar{\hat{\mathbf{m}}}_{k,j}^{[p]}\right)\right), \tag{B.14}$$

where $\mathbf{K}_k$ is the Kalman gain matrix, $\mathbf{z}_k$ is the measurement and $\mathbf{h}(\cdot)$ is the measurement model defined in Appendix A (A.16). The Kalman gain matrix behaves as a weighing matrix that determines how much the estimate should be corrected using the measurement. The Kalman gain matrix is defined as

$$\mathbf{K}_{k,j}^{[p]} = \bar{\boldsymbol{\Sigma}}_{k,j}^{[p]}\mathbf{H}_{k,j}^{\mathrm{T}}\left(\mathbf{H}_{k,j}\bar{\boldsymbol{\Sigma}}_{k,j}^{[p]}\mathbf{H}_{k,j}^{\mathrm{T}} + \mathbf{Q}_k\right)^{-1}, \tag{B.15}$$

where $\mathbf{H}_k$ is the Jacobian of $\mathbf{h}(\cdot)$ with respect to the state. Using the previously defined measurement model

$$\begin{aligned} \mathbf{H}_k &= \left.\frac{\partial h}{\partial x}\right|_{\bar{\hat{\mathbf{x}}}_k} \\ &= \begin{bmatrix} \frac{\bar{\hat{x}}_{k,j}^{[p]} - x_k^{[p]}}{\sqrt{q}} & \frac{\bar{\hat{y}}_{k,j}^{[p]} - y_k^{[p]}}{\sqrt{q}} \\ \frac{\mathbf{y}_k^{[p]} - \hat{\hat{y}}_{k,j}^{[p]}}{q} & \frac{\bar{\hat{x}}_k^{[p]} - x_k^{[p]}}{q} \end{bmatrix} \end{aligned} \tag{B.16}$$

where $q \triangleq \left(\bar{\hat{y}}_{k,j}^{[p]} - y_k^{[p]}\right)^2 + \left(\bar{\hat{x}}_{k,j}^{[p]} - x_k^{[p]}\right)^2$, and the covariance matrix of the measurement model is defined as

$$\mathbf{Q}_k = \begin{bmatrix} \sigma_{k,r}^2 & 0 \\ 0 & \sigma_{k,\phi}^2 \end{bmatrix}, \tag{B.17}$$

where $\sigma_{k,r} \in \mathbb{R}$ and $\sigma_{k,\phi} \in \mathbb{R}$ are the standard deviation of the range and bearing measurements respectively.

The final step of the feature estimate update is correction of the estimate's covariance matrix. The covariance matrix is corrected according to

$$\mathbf{\Sigma}_{k,j} = (\mathbf{I} - \mathbf{K}_{k,j}\mathbf{H}_{k,j})\overline{\mathbf{\Sigma}}_{k,j}, \tag{B.18}$$

where $\mathbf{I}$ is an identity matrix of size 2. Once each of the landmark estimates have been updated using the current set of observations, the third step of the FastSLAM algorithm is the importance weight calculation the details of which are presented in the following section.

B.4. Importance Weight Calculation

As discussed in Chapter 6 we are sampling particles from a distribution that does not match the distribution that we are attempting to estimate. From [77] if the importance weight of each particle is generated as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \tag{B.19}$$

a particles are then drawn with replacement from $\overline{\mathcal{X}}_k$ and added to $\mathcal{X}_k$ then the estimate produced by the Rao-Blackwellized particle filter will approximate the target distribution and the quality of the estimate will increase as the number of particles increases. As shown in Chapter 6 Section 6.2.3 the importance weight of each particle can be calculated in closed form as

$$w_k^{[p]} = \eta \prod_{j=1}^{m} \left| 2\pi \mathbf{Q}_{i,t}^{[m]} \right| \exp\left\{ -\frac{1}{2} \left(\mathbf{z}_k^j - h\left(\mathbf{m}_i^{[m]} \right) \right)^{\mathrm{T}} \frac{1}{\mathbf{Q}_{i,t}^{[m]}} \left(\mathbf{z}_k^j - h\left(\mathbf{m}_i^{[m]} \right) \right) \right\} \tag{B.20}$$

where the covariance matrix $\mathbf{Q}_{i,t}^{[m]}$ is defined as

$$\mathbf{Q}_{i,t}^{[m]} = \mathbf{H}_k^{[m]\mathrm{T}} \mathbf{\Sigma}_{i,t-1}^{[m]} \mathbf{H}_k^{[m]} + \mathbf{Q}_k, \tag{B.21}$$

where $\mathbf{H}_{k,j}$ is the same Jacobian described in Section B.3.2 and $\mathbf{Q}_k$ is the covariance matrix of the measurement model.

B.5. Resampling

The resampling step takes the set of potential particles which are distributed according to the proposal distribution and transforms the particle set so that it approximates the target distribution. This transformation occurs by sampling with replacement from the set of potential particles according to the particles importance weight, that is particles with larger importance weights have a higher probability of being selected in the sampling process. The resampling algorithm chosen for comes from [53] and is referred to as a **low_variance_sampler**. The pseudocode for the sampler is given in Algorithm B.4.

Algorithm B.4 Overview of the low variance sampler algorithm.

procedure LowVarianceSampler($\overline{\mathcal{X}}_k$)
  Data:
    $\overline{\mathcal{X}}_k$ the set of potential particles, of length n

  Result:
    $\mathcal{X}_k$ the final set of particles after resampling

  $\mathcal{X}_k \leftarrow \{\}$
  $r = \text{rand}\,(0; n^{-1})$
  $c = w_t^{[1]}$
  $i = 1$
  for $m = 1$ to n **do**
    $U = r + (m-1) \cdot M^{-1}$
    while $U > c$ **do**
      $i = i + 1$
      $c = c + w_t^{[i]}$
    end while
    $\mathcal{X}_k^{[i]} \leftarrow \overline{\mathcal{X}}_k^{[i]}$
  end for
end procedure

B.6. New Feature Addition

The final component of the FastSLAM algorithm that must be addressed is the addition of new landmarks to the map maintained by each particle. When a new landmark, whose location is not currently being estimated by a particle, is encountered the EKF that estimates the landmark position must be initialized. This initialization is based on the inverse of the measurement model

$$\hat{\mathbf{m}}_{k,n+1}^{[p]} = \mathbf{f}\left(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k\right), \tag{B.22}$$

which calculates the position of the landmark based on the particle pose and the observation that corresponds to the newly observed landmark. For our system the inverse measurement model is defined as

$$\hat{\mathbf{m}}_{k,n+1}^{[p]} = \mathbf{f}\left(\boldsymbol{\xi}_k^{[p]}, \mathbf{z}_k(i)\right) = \begin{bmatrix} \hat{x}_{k,n+1}^{[p]} \\ \hat{y}_{k,n+1}^{[p]} \end{bmatrix} = \begin{bmatrix} x_k^{[p]} + r_k(i)\cos\left(\theta_k^{[p]} - \phi_k(i)\right) \\ y_k^{[p]} + r_k(i)\sin\left(\theta_k^{[p]} - \phi_k(i)\right) \end{bmatrix}. \tag{B.23}$$

The initialization of the EKF covariance matrix, that represents the uncertainty in the landmark position estimate, is the final step of the new landmark addition. The initialization of the covariance matrix for the FastSLAM algorithm is slightly different than the initialization of the covariance matrix for the EKF SLAM algorithm. In the EKF SLAM algorithm the initialization of the covariance matrix for newly observed features is a function of the uncertainty in the robots pose estimate along with the uncertainty of the feature observation. However, in contrast the uncertainty in the robots pose estimate in the FastSLAM algorithm is represented in the spread of the particle set. The result is that the initialization of the covariance matrix of the feature estimate is solely a function of the uncertainty of the feature observation. The resulting covariance initialization is performed according to

$$\boldsymbol{\Sigma}_{k,n+1}^{[p]} = \mathbf{F}_{k,z}\mathbf{Q}_k\mathbf{F}_{k,z}^{\mathrm{T}}, \tag{B.24}$$

which transforms the uncertainty of the measurement, characterized by the measurement

covariance matrix $\mathbf{Q}_k$, into the global frame of reference used for the feature location estimate. The matrix, $\mathbf{F}_{k,z}$, is the Jacobian of $\mathbf{f}(\cdot)$ with respect to the observation, which for the previously described inverse measurement model is defined as

$$\begin{aligned}\mathbf{F}_{k,z} &= \left.\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right| \\ &= \begin{bmatrix} \cos\left(\theta_k^{[p]} - \phi_k(i)\right) & r_k(i)\sin\left(\theta_k^{[p]} - \phi_k(i)\right) \\ \sin\left(\theta_k^{[p]} - \phi_k(i)\right) & -r_k(i)\cos\left(\theta_k^{[p]} - \phi_k(i)\right) \end{bmatrix}. \end{aligned} \tag{B.25}$$

With all of the details of the FastSLAM algorithm described the following chapter contains the details of the extension of the FastSLAM algorithm that represents the environment in which the UMV operates using an occupancy grid.

Appendix C

FastSLAM OG Details

C.1. Introduction

As discussed in Chapter 7 the FastSLAM OG algorithm extends the featured based FastSLAM algorithm by replacing the map, composed of a set of landmarks, with an occupancy grid. The FastSLAM OG algorithm estimates the distribution that represents the trajectory of the UMV $\boldsymbol{\xi}_{1:k}$ and the map $\mathbf{M}$ based on the set of control inputs $\mathbf{u}_{1:k}$ and observations $\mathbf{z}_{1:k}$. In a probabilistic sense this distribution is given as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right). \tag{C.1}$$

FastSLAM OG then factors the joint distribution into a pair of distributions that are simpler to estimate, the factored form of the distribution is given as

$$p\left(\boldsymbol{\xi}_{1:k}, \mathbf{M} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) = p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) p\left(\mathbf{M} \mid \boldsymbol{\xi}_{1:k}, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right), \tag{C.2}$$

where $p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right)$ is estimated using a Rao-Blackwellized particle filter and each particle filter maintains its own estimate of the map. In FastSLAM OG the map is represented using an occupancy grid, thus the pth particle in the particle filter is defined as

$$\mathcal{X}_k^{[p]} \triangleq \begin{bmatrix} \boldsymbol{\xi}_k^{[p]} & \mathbf{M}_k^{[p]} \end{bmatrix}, \tag{C.3}$$

where $\boldsymbol{\xi}_k^{[p]}$ is the pose estimate of the particle and $\mathbf{M}_k^{[p]}$ is the occupancy grid of the particle. The overview of the FastSLAM OG algorithm is presented in Chapter 7 Section 7.3 however it is displayed again in Algorithm C.1 for easy reference.

Algorithm C.1 Overview of the FastSLAM OG algorithm repeated for reference.

procedure FastSLAMOG($\mathcal{X}_{k-1}$, $\mathbf{u}_k$, $\mathbf{z}_k$)
    Data:
        $\mathcal{X}_{k-1}$ the particle set from the previous time step
        $\mathbf{u}_k$ the current control input
        $\mathbf{z}_k$ the current set of measurements

    Result:
        $\mathcal{X}_k$ the particle set from the current time step

    $\overline{\mathcal{X}}_k \leftarrow \{\}$
    for $p \leftarrow 1, n$ **do**
        $\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right)$
        $w^{[p]} \leftarrow \eta p\left(\mathbf{z}_k \,\middle|\, \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$
        $\mathbf{M}_k^{[p]} = \text{OGUpdate}(\boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}, \mathbf{z}_k)$
        $\overline{\mathcal{X}}_k^{[p]} \leftarrow \begin{bmatrix} \boldsymbol{\xi}_k^{[p]} & \mathbf{M}_k^{[p]} & w^{[p]} \end{bmatrix}$
    end for
    $\mathcal{X}_k \leftarrow \{\}$
    for $p = 1, n$ **do**
        select j with probability proportional to $w^{[p]}$
        $\mathcal{X}_k^{[p]} \leftarrow \overline{\mathcal{X}}_k^{[j]}$
    end for
end procedure

C.1.1. Pose Sampling

In the first step of the FastSLAM OG algorithm a set of potential particles are generated. The pose of the pth particle in the particle filter is generated by sampling from the probabilistic motion model of the UMV,

$$\boldsymbol{\xi}_k^{[p]} \sim p\left(\boldsymbol{\xi}_k \,\middle|\, \boldsymbol{\xi}_{k-1}^{[p]}, \mathbf{u}_k\right), \tag{C.4}$$

where $\boldsymbol{\xi}_k^{[p]}$ is the pose that is generated through the sampling process, $\boldsymbol{\xi}_{k-1}^{[p]}$ is the pose from the previous time step on which the new pose is based, and $\mathbf{u}_k$ is the current control input. For the FastSLAM OG algorithm the pose sampling is performed using the same procedure using by the feature based version of FastSLAM, the details of which can be seen in Appendix B Section B.2. The result of the sampling process is a set of particles that are distributed according to the proposal distribution, $p\left(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right)$. After the potential particles have been generated the second step of the algorithm incorporates the current measurement, $\mathbf{z}_k$, into the estimate through the importance weight calculation, the details of which are presented in the following section.

C.2. Importance Weight Calculation

The second step of the algorithm is to calculate the importance weight for each particle. We know from Chapter 7 Section 7.3.2 that the importance weight of each particle is given as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, = \eta p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right), \tag{C.5}$$

where $\eta > 0$ is a normalization factor and $p\left(\mathbf{z}_k \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right)$ is the probabilistic measurement model of the sensor. The probabilistic sensor model calculates the probability that the current observation $\mathbf{z}_k$ would be made based on the particle pose estimate $\boldsymbol{\xi}_k^{[p]}$ and the occupancy grid currently maintained by the particle $\mathbf{M}_{k-1}^{[p]}$.

For the previously discussed custom laser based rangefinder (Chapter 3) it is appropriate to use the beam model developed in [53]. The beam model takes into account four different sources of noise for a given range measurement when generating the probability that a sensor measurement would be made. The first source of error is referred to as simple measurement noise and is modeled as a Gaussian distribution represented by a mean whose value is $z_k^\star$ and standard deviation σ_h. The mean value representing the Gaussian, $z_k^\star$, represents the

true value of the observation if a perfect sensor was used. In the probabilistic measurement model this perfect sensor value is calculated using the pose of the particle, $\boldsymbol{\xi}_k^{[p]}$, and the particle's occupancy grid, $\mathbf{M}_k^{[p]}$, and the bearing that corresponds to the ith observation, $\phi_k(i)$, in $\mathbf{z}_k$. Based on the pose of the particle and the bearing of the measurement a line is drawn from the location of the particle out in direction of the measurement. Each occupancy grid cell that the line transverses is examined and the first occupied cell, that is a cell in the occupancy grid that has a probability of occupancy greater than a minimum threshold of occupancy $p\left(\mathbf{M}_{k-1}^{[p]}(i,j)\right) \geq p_o$, is used to calculate the true value of the measurement based on the euclidian distance between the vehicle pose and the center of the occupied cell, this approach is referred to as ray tracing. Using the ideal sensor measurement generated from the occupancy grid and particle pose the probability that a sensor observation would be made due to effect of sensor noise is given as

$$p_h\left(\mathbf{z}_k(i) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_{k-1}^{[p]}\right) = \begin{cases} \eta\mathcal{N}\left(z_k^\star, \sigma_h^2\right) & \text{if } 0 \leq \mathbf{z}_k(i) \leq z_{max} \\ 0 & \mathbf{z}_k(i) > z_{max} \end{cases} \tag{C.6}$$

where z_{max} is the maximum range of the sensor, and $\mathcal{N}(\cdot,\cdot)$ is the standard close form of a Gaussian distribution defined as

$$\mathcal{N}\left(z_k^\star, \sigma_h^2\right) = \frac{1}{\sqrt{2\pi\sigma_h^2}} \exp\left(-\frac{1}{2}\frac{\left(\mathbf{z}_k(i) - z_k^\star\right)^2}{\sigma_h^2}\right), \tag{C.7}$$

where $\eta > 0$ is a normalizing term. For a Gaussian distribution the normalizer can be calculated in closed form as

$$\eta = \left(\int_0^{z_{max}} \frac{1}{\sqrt{2\pi\sigma_h^2}} \exp\left(-\frac{1}{2}\frac{\left(\mathbf{z}_k(i) - z_k^\star\right)^2}{\sigma_{hit}^2}\right) d\mathbf{z}_k(i)\right)^{-1}. \tag{C.8}$$

The second source of error that exists in the beam model is produced by returns generated due to the existence of unexpected objects in the environment. A common source of these unexpected measurements is dynamic objects in the robot's environment. As previously discussed, we assume that the environment that the UMV is operating is static, thus any

dynamic objects in the environment will produced unexpected measurements when compared to what is expected based on the occupancy grid. The existence of unexpected objects is modelled as an exponential based on the assumption that the probability of measuring an unexpected object decreases the farther that you get from the sensor. The resulting probability of measuring an unexpected object is

$$p_x\left(\mathbf{z}_k(i) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_k^{[p]}\right) = \begin{cases} \eta\lambda_s \exp\left(-\lambda_s \mathbf{z}_k(i)\right) & 0 \leq \mathbf{z}_k(i) \leq z_k^\star \\ 0 & \mathbf{z}_k(i) > z_k^\star \end{cases} \tag{C.9}$$

where λ_s is a constant that defines the exponential and $\eta > 0$ is a normalizing constant. The normalizer for an exponential distribution can be calculated in closed form according to

$$\eta = \left(\int_0^{z_k^\star} \lambda_s \exp\left(-\lambda_s \mathbf{z}_k(i)\right)\right)^{-1} \tag{C.10}$$

$$= \left(-\exp\left(-\lambda_s z_k^\star\right) + \exp\left(-\lambda_s\right)\right)^{-1} \tag{C.11}$$

$$= \left(1 - \exp\left(\lambda_s z_k^\star\right)\right)^{-1}. \tag{C.12}$$

The third source of error comes from miscalculations that are caused by the misidentification of the laser lines in the range sensor. When that happens the range returned by the sensor is the sensors maximum range, thus these errors are modelled as a point mass at the sensors maximum range. The probability that the sensor miscalculates the laser lines is given as

$$p_m\left(\mathbf{z}_k(i) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_k^{[p]}\right) = \begin{cases} 1 & \mathbf{z}_k(i) = z_{max} \\ 0 & \mathbf{z}_k(i) \neq z_{max}. \end{cases} \tag{C.13}$$

Finally, errors produced by random miscalculations are included. These errors are modelled as a uniform distribution over the entire sensor range whose provability is inversely proportional to the maximum range of the sensor. The probability of a sensor measurement being produced by random errors is given is

$$p_r\left(\mathbf{z}_k(i) \;\middle|\; \boldsymbol{\xi}_k^{[p]}, \mathbf{M}_k^{[p]}\right) = \begin{cases} \frac{1}{z_{max}} & 0 \leq \mathbf{z}_k(i) \leq z_{max} \\ 0 & \mathbf{z}_k(i) > z_{max}. \end{cases} \tag{C.14}$$

With the four possible sources of error calculated, each of the sources of error are mixed using a set of mixing parameters, $z_h + z_s + z_m + z_r = 1$, which determine how much of an effect each source of error has in the probabilistic measurement model, an example is shown in Figure C.1. Finally, the importance weight of a given particle is calculated as

$$w_t^{[k]} = \prod_k \begin{bmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{random}} \end{bmatrix}^{\text{T}} \begin{bmatrix} p_{hit}\left(z_t^k | x_t^{[k]}, M^{[k]}\right) \\ p_{short}\left(z_t^k | x_t^{[k]}, M^{[k]}\right) \\ p_{max}\left(z_t^k | x_t^{[k]}, M^{[k]}\right) \\ p_{random}\left(z_t^k | x_t^{[k]}, M^{[k]}\right) \end{bmatrix}. \tag{C.15}$$

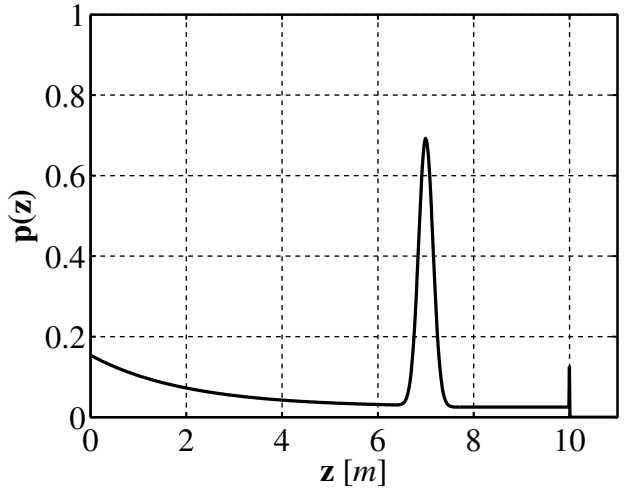


Figure C.1: Beam measurement model used by the laser based rangefinder used for the importance weight update.

C.3. Occupancy Grid Map Update

The third step of the FastSLAM OG algorithm is the update of the occupancy grid maintained by each particle. An overview of the occupancy grid update algorithm was presented in Chapter 7 Section 7.2 but it is restated in Algorithm C.2 for reference. The key component of the occupancy grid update algorithm is the inverse sensor model, $p\left(\mathbf{M}\left(i,j\right) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l)\right)$, which calculates the probability that a cell will be occupied based on the current sensor

measurement $\mathbf{z}_k$ and the pose of the particle $\boldsymbol{\xi}_k^{[p]}$. The details of the inverse sensor model that was chosen are provided in the following section.

Algorithm C.2 Overview of the standard occupancy grid update algorithm repeated for reference.

procedure OGUPDATE($\boldsymbol{\xi}_k$,$\mathbf{M}_{k-1}$, $\mathbf{z}_k$)
    Data:
        $\boldsymbol{\xi}_k$ the current UMV pose
        $\mathbf{M}_{k-1}$ the occupancy grid from the previous time step, of size $w \times h$
        $\mathbf{z}_k$ the current sensor measurement, of length m

    Result:
        $\mathbf{M}_k$ the occupancy grid for the current time step

    for $i = 1, h$ **do**
        for $j = 1, w$ **do**
            for $l = 1, m$ **do**
                if $\mathbf{M}_{k-1}(i,j)$ lies in the perceptual range of $\mathbf{z}_k(l)$ **then**
                    $\mathbf{M}_k(i,j) \leftarrow \mathbf{M}_{k-1}(i,j) + \log \frac{p(\mathbf{M}(i,j) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l))}{1-p(\mathbf{M}(i,j) \mid \boldsymbol{\xi}_k, \mathbf{z}_k(l))} - l_0$
                end if
            end for
        end for
    end for
end procedure

C.3.1. Inverse Sensor Model

The inverse sensor model that was chosen consists of three specific regions that represent the three physical regions of a given distance measurement. For the laser based range finder developed in Chapter 3 the inverse sensor model refers to each distance measurement at a given relative bearing as a beam. For each beam the model uses three separate regions to characterize the probability of occupancy for a given cell. Along the beam before an object is encountered, that is when the distance along the beam is less than the measured distance, is referred to as *free space*. At the measured distance along the beam the cell in the environment, based on the distance measurement and pose of the vehicle, is what is referred

to as *occupied*. Finally, locations beyond the measured distance along the beam are referred to as *unknown space*. The result of the regional breakdown is that the inverse sensor model returns the probability of occupancy of a cell based on the region that the cell falls into.

For the *free space* region we assume a linear increase in probability from $p_{f,l}$ at a measured distance $r_i = 0$ to $p_{f,h}$ at $r_i = z_{max}$. A linear relationship is used as opposed to a constant because as shown in the sensor design section the error increases with range, thus the confidence in the measurement decreases with distance which leads to an increase in occupancy probability. In the *occupied space* we also assume a linear relationship from $p_{o,h}$ at $r = r_i - \gamma_r$ to $p_{o,l}$ at $r = r_i + \gamma_r$. The bounding window from $r_i - \gamma_r$ to $r_i + \gamma_r$ is used to account for the uncertainty in the distance measurement. Finally, in the *unknown space* region, $r > r_i$ a value of $p\left(\mathbf{M}_k^{[p]}(i,j)\right)$ which is the prior of occupancy is returned because no information is known about the cells. There is one other region that is also taken into account, that is when $r_i = z_{max}$ or when the sensor returns max range readings. Through the design of the sensor max ranges are returned when there is a misidentification of the two laser lines or when invalid ranges are calculated. Thus, when a max range is returned by the sensor the inverse sensor model returns a value of $p\left(\mathbf{M}_k^{[p]}(i,j)\right)$ for all cells along the beam because no valid information is known for that beam. A plot of the inverse sensor model is given in Figure C.2 which is used by the occupancy grid update algorithm.

C.4. Resampling

The final step in the occupancy grid based FastSLAM algorithm is to resample the particles. The resampling process takes the set of particles that were sampled and updated based on the proposal distribution $p(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{k-1})$ and ensures that the final particle set, $\mathcal{X}_k$, produced by the resampling step estimates the target distribution, $p(\boldsymbol{\xi}_{1:k} \mid \mathbf{u}_{1:k}, \mathbf{z}_{1:k})$. For resampling, the same simple resampling process described previously in Appendix B

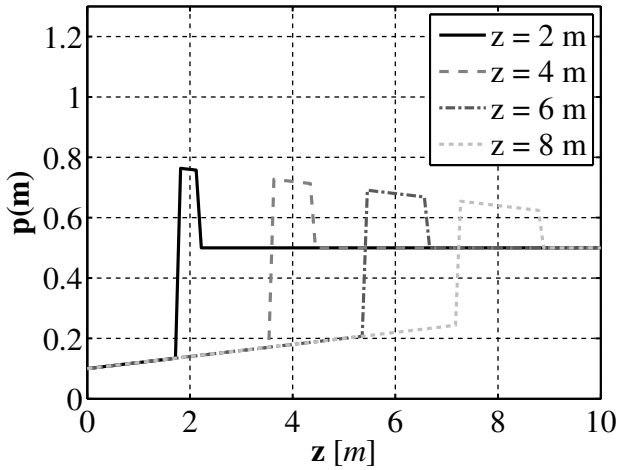


Figure C.2: Inverse sensor model used for the developed laser based rangefinder.

Section B.5 is used.