

ABSTRACT

Title of Document: THEORY, DESIGN, AND IMPLEMENTATION OF
LANDMARK PROMOTION COOPERATIVE
SIMULTANEOUS LOCALIZATION AND MAPPING

John George Karvounis, Doctor of Philosophy, 2011

Dissertation Directed By: Professor Gilmer Blankenship
Department of Electrical and Computer Engineering

Simultaneous Localization and Mapping (SLAM) is a challenging problem in practice, the use of multiple robots and inexpensive sensors poses even more demands on the designer. Cooperative SLAM poses specific challenges in the areas of computational efficiency, software/network performance, and robustness to errors. New methods in image processing, recursive filtering, and SLAM have been developed to implement practical algorithms for cooperative SLAM on a set of inexpensive robots.

The Consolidated Unscented Mixed Recursive Filter (CUMRF) is designed to handle non-linear systems with non-Gaussian noise. This is accomplished using the Unscented Transform combined with Gaussian Mixture Models. The Robust Kalman Filter is an extension of the Kalman Filter algorithm that improves the ability to remove

erroneous observations using Principal Component Analysis (PCA) and the X84 outlier rejection rule. Forgetful SLAM is a local SLAM technique that runs in nearly constant time relative to the number of visible landmarks and improves poor performing sensors through sensor fusion and outlier rejection. Forgetful SLAM correlates all measured observations, but stops the state from growing over time. Hierarchical Active Ripple SLAM (HAR-SLAM) is a new SLAM architecture that breaks the traditional state space of SLAM into a chain of smaller state spaces, allowing multiple robots, multiple sensors, and multiple updates to occur in linear time with linear storage with respect to the number of robots, landmarks, and robots poses. This dissertation presents explicit methods for closing-the-loop, joining multiple robots, and active updates. Landmark Promotion SLAM is a hierarchy of new SLAM methods, using the Robust Kalman Filter, Forgetful SLAM, and HAR-SLAM.

Practical aspects of SLAM are a focus of this dissertation. LK-SURF is a new image processing technique that combines Lucas-Kanade feature tracking with Speeded-Up Robust Features to perform spatial and temporal tracking. Typical stereo correspondence techniques fail at providing descriptors for features, or fail at temporal tracking. Several calibration and modeling techniques are also covered, including calibrating stereo cameras, aligning stereo cameras to an inertial system, and making neural net system models. These methods are important to improve the quality of the data and images acquired for the SLAM process.

THEORY, DESIGN, AND IMPLEMENTATION OF LANDMARK
PROMOTION COOPERATIVE SIMULTANEOUS LOCALIZATION
AND MAPPING

By

John George Karvounis

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:

Professor Gilmer Blankenship, Chair
Professor Christopher Davis
Professor Nuno Martins
Professor Steven Marcus
Professor David Jacobs

© Copyright by
John George Karvounis
2011

DEDICATION

To my parents,
who always believed in me and asked me to do my best.

ACKNOWLEDGEMENTS

I would like to acknowledge a great many people who have helped me on my journey. First, I want to thank my advisor, Professor Gilmer Blankenship for all his support and stimulating thoughts. He allowed me to find my own path throughout this, and was always supportive of my methods.

My deepest gratitude goes toward my friends at the Autonomous Systems Lab. A special thanks to Jared Napora and Michael Stanley for all their help over the years. Together we designed robots, software, and kept each other sane over the years. We all started as graduate students together, and ended up as close friends.

I must thank TRX Systems as a whole, not only as a great source of information and technology, but as a great work environment as well. Exceptional thanks goes to Dr. Carole Teolis, who went above and beyond as a boss and was a surrogate advisor to me throughout the process. An additional thanks to Amrit Bandyopadhyay, who was always supportive of my work and always there to bounce ideas.

Of course I cannot forget my family, they always believed in me and gave me all the confidence I needed to complete this journey. My family always encouraged me to do my best.

TABLE OF CONTENTS

Dedication.....	ii
Acknowledgements.....	iii
Table of Contents.....	iv
List of Tables.....	viii
List of Figures.....	x
Chapter 1: Introduction.....	1
1.1. Introduction to SLAM.....	1
1.2. Problem Definition.....	4
1.2.1. Purpose.....	5
1.2.2. Challenges.....	5
1.3. Hardware Selection.....	7
1.4. Solution.....	7
Chapter 2: Literature Review.....	10
2.1. Fusing Inertial and Visual Sensors.....	10
2.1.1. Visual Odometry.....	11
2.1.2. Inertial Navigation.....	13
2.1.3. Inspiration from Biology.....	14
2.1.4. Examples of Fused Data.....	17
2.2. Simultaneous Localization and Mapping (SLAM).....	20
2.2.1. Kalman Filters.....	24
2.2.1.1. Basic Kalman Filter.....	25
2.2.1.2. Extended Kalman Filter.....	27
2.2.1.3. Unscented Kalman Filter.....	31
2.2.2. Information Filter.....	34
2.2.2.1. Sparse Extended Information Filter.....	35
2.2.2.2. GraphSLAM.....	36
2.2.3. Particle Filters.....	38
2.2.3.1. Rao-Blackwellized Particle Filter.....	39
2.2.4. FastSLAM 1.0 and 2.0.....	41
2.2.5. Monocular SLAM.....	42
2.2.6. Stereo Vision SLAM.....	45
2.2.6.1. Stereo SIFT SLAM.....	46
2.3. Cooperative Control and Coverage.....	48
2.3.1. Cellular Decomposition.....	48
2.3.1.1. Spanning Trees.....	49
2.3.2. Server-based Swarm.....	50

2.3.3.	Pseudo-Potential Fields.....	50
2.3.4.	Graph Coordination	53
2.3.5.	Optimal Cooperative Coverage.....	54
2.4.	Landmark and Feature Analysis	55
2.4.1.	Scale Invariant Feature Transform (SIFT).....	56
2.4.2.	Speeded Up Robust Features (SURF).....	57
2.4.3.	Harris Corners.....	58
2.4.4.	Lucas-Kanade feature tracker	61
2.5.	Literature Review Remarks	64
Chapter 3:	LK-SURF	66
3.1.	Stereo Correspondence	67
3.2.	Correspondence Over Time.....	68
3.3.	Proposed Solution.....	69
3.4.	Advantages of LK-SURF	71
3.5.	Disadvantages of LK-SURF.....	73
3.6.	Applications.....	74
Chapter 4:	CUMRF	75
4.1.	Gaussian Mixture Models.....	77
4.2.	Merging Sigma Points	79
4.3.	Filter Details	80
4.4.	Example	83
4.5.	Applications.....	85
Chapter 5:	Robust Kalman Filter	87
5.1.	Removing Outliers from the Kalman Filter.....	89
5.2.	Adaptations of the Robust Kalman Filter	94
5.3.	Example	95
5.4.	Analysis	96
5.5.	Applications.....	100
Chapter 6:	Forgetful SLAM	101
6.1.	Method Details.....	102
6.2.	Advantages of Forgetful SLAM	108
6.3.	Disadvantage of Forgetful SLAM	110
6.4.	Modification for Neural Net Based Models	110
6.5.	Applications.....	112
Chapter 7:	Hierarchical Active Ripple SLAM.....	113
7.1.	Modifications of Forgetful SLAM for HAR-SLAM.....	115
7.2.	Method Details.....	120
7.2.1.	Standard Update.....	121
7.2.2.	Revisiting Landmark Update	123
7.2.3.	Multiple Robot Updates.....	125
7.2.4.	Derivation of HAR-SLAM Updates	131
7.3.	Advantage of HAR-SLAM.....	136

7.4.	Disadvantages of HAR-SLAM.....	138
7.5.	Computational Analysis.....	139
7.6.	Applications.....	141
Chapter 8:	Cooperative SLAM with Landmark Promotion	142
8.1.	Method Details.....	142
8.1.1.	Storing.....	145
8.1.1.1.	Landmark Promotion Metric.....	146
8.1.2.	Matching	146
8.1.3.	Map merging.....	149
8.2.	Related Topics	150
8.2.1.	Cooperative Control.....	151
8.2.2.	Path finding.....	151
8.2.3.	Central Server Map Manager.....	152
8.3.	Examples.....	153
Chapter 9:	Hardware and Design	160
9.1.	Building the Robot.....	160
9.1.1.	Sensor Board.....	163
9.1.2.	Optical INU.....	164
9.1.3.	Computing Platform.....	165
9.1.4.	Mobile Mesh Network	166
9.2.	Programming the Robot.....	167
9.2.1.	Event Architecture	171
9.2.2.	Platform.....	171
9.2.3.	Algorithms	173
9.2.4.	Networking	174
9.3.	Software Components Developed for Cooperative SLAM.....	175
9.3.1.	Tamiya Platform	176
9.3.2.	Landmark Promotion SLAM Algorithm	180
Chapter 10:	Calibration and Modeling.....	187
10.1.	Encoder Calibration and Error Modeling	187
10.2.	Stereo Camera Calibration and Error Modeling	189
10.2.1.	Calculating Camera Calibration.....	189
10.2.2.	Rectifying Stereo Images.....	191
10.2.3.	Modeling Error.....	193
10.3.	Inertial Error Modeling.....	196
10.4.	Aligning Optical and Inertial Systems.....	198
10.5.	Robot System Modeling	201
10.5.1.	Comparing linear vs. nonlinear models	204
10.6.	Robot State and Observation Models	206
Chapter 11:	Conclusion.....	211
11.1.	Problem Statement.....	211
11.2.	Solution to Problem Statement	211
11.3.	Theoretical Advances	212

11.4. Implementation Advances	214
11.5. Final Remarks	215
Chapter 12: Future Work.....	217
12.1. Heterogeneous Robots Using the Map	217
12.2. Real-time Tracking Using GPU Technology	217
12.2.1. Image Processing on the GPU	218
12.2.2. Linear Algebra on the GPU	219
12.2.3. Identifying Other Robots in Real-time	220
12.3. Pedestrian Tracking	221
12.3.1. Improve Error Models and Model Human Motion.....	221
Appendix A: Useful EMGU Examples.....	224
Stereo Camera Calibration.....	224
Stereo Rectifying	225
Stereo Triangulation with Epipolar Geometry	226
Appendix B: Matlab Implementations of Various Kalman Filters.....	229
Basic Kalman Filter	229
Extended Kalman Filter	230
Unscented Kalman Filter	232
Gaussian Consolidation	235
List of References	236

LIST OF TABLES

Table 1 - Averages and standard deviations for various conditions and speeds [17]	16
Table 2 - Basic Kalman Filter equations [32].....	26
Table 3 - Extended Kalman Filter equations [45].....	28
Table 4 - Unscented Kalman Filter equations [27][49]	33
Table 5 - Basic Information Filter equations [27]	34
Table 6 - Particle Filter propagation of a Gaussian [27].....	39
Table 7 - Pseudo-potential equations for maintaining connectivity and collision avoidance [65].....	53
Table 8 - Derivation of the Harris Corner Detector [23]	59
Table 9 - Outline of the LK Tracker [24]	63
Table 10 - Pseudo-code for sub-pixel calculation [24].....	64
Table 11 - Outline of LK-SURF algorithm.....	70
Table 12 - Algorithm for merging Gaussians using sigma points	80
Table 13 - Example equations.....	83
Table 14 - Observation weighting for removal.....	90
Table 15 - Outlier removal for single dimensional arrays	91
Table 16 - Robust Kalman Filter algorithm	93
Table 17 - Filter performance comparison using percent error over distance travelled. Eight paths used to compare inertial paths, Robust Kalman Filter paths, and Extended Kalman Filter paths.	97
Table 18 - Plot snapshots of inertial paths, Robust Kalman Filter paths, and EKF paths.....	99
Table 19 - System equations and notation	103
Table 20 - Algorithm for removing lost features	104
Table 21 - Algorithm for adding new features.....	105
Table 22 - Algorithm of Forgetful SLAM	107
Table 23 - State prediction using Unscented Transform	111

Table 24 - Matrix notation for feature removal	116
Table 25 - Algorithm for retaining lost features	117
Table 26- Augmented state functions for pose correlation.....	118
Table 27 - Algorithm for creating pose cross variance from augmented state	119
Table 28 - Reduction of the augmented state.....	120
Table 29 - Algorithm to update past poses and landmarks.....	122
Table 30 - Algorithm to update past covariance matrices	122
Table 31 - Algorithm to update revisited landmarks	124
Table 32 - Algorithm to update map links after removing a revisited landmark	125
Table 33 - Kalman Filter for global coordinate update.....	127
Table 34 - Robust global coordinate update	129
Table 35 - Landmark update between multiple robots	130
Table 36 - Verification that gain selection in link update results in source state matching observation state	132
Table 37 - Verification that gain selection in link update results in source covariance matching observation.....	133
Table 38 - Verification that using an observation covariance equal to the source covariance in link update has no effect on the source covariance, destination covariance, or cross-covariance	134
Table 39 - Verification of gain selection in update revisited landmark.....	136
Table 40 - Comparison of SLAM computational complexity	140
Table 41 - Calculating landmark transformation	147
Table 42 - Calculation of landmark fitness.....	148
Table 43 - Test platform components	162
Table 44 - Kabsch algorithm [95].....	200
Table 45 - Robot and map state	207
Table 46 - Quaternion functions	207
Table 47 - Pose prediction function and Jacobian matrix.....	208
Table 48 - Robot and map prediction function and Jacobian matrix	208
Table 49 - Observation prediction functions	209

LIST OF FIGURES

Figure 1 - The essential SLAM problem [1].....	2
Figure 2 - The effect of gyroscope drift on mapping [16]	13
Figure 3 - Mean angular velocity vs. physical displacement [17]	16
Figure 4 - Three frames of reference for perceived orientation of a test line with respect to gravity [19]	17
Figure 5 - 3D reconstruction algorithm [20].....	18
Figure 6 - Stereo image with detected edges and vertical lines [20]	18
Figure 7 - The epipolar constraint, satisfied by ideal sensor data [26]	20
Figure 8 - Drift in odometry [27].....	21
Figure 9 - SLAM model using a spring network analogy [1].....	23
Figure 10 - Linear transformation of probability [27]	27
Figure 11 - Example of linearization for Gaussian propagation [45]	29
Figure 12 - Comparison of linearized Gaussian propagation vs. true propagation [27].....	29
Figure 13 - The convergence in landmark uncertainty [1]	31
Figure 14 - Example of Unscented Transform with sigma points for Gaussian propagation [45]	32
Figure 15 - Comparison of Unscented Transform with sigma points for Gaussian propagation vs. true propagation [27]	32
Figure 16 - Graphical representation of sparsification from Information Matrix to map [27].....	36
Figure 17 - Sparsification with sub-maps. Maps (a) and (b) are combined to form map (c) [50].....	36
Figure 18 - Conceptual diagram of GraphSLAM constraints in the Information Matrix [51].....	37
Figure 19 - Conceptual diagram of how GraphSLAM removes landmarks [51].....	37
Figure 20 - Sample robot trajectory using the FastSLAM approach [1]	42
Figure 21 - Sample snapshots and projections from MonoSLAM [46].....	43
Figure 22 - Boustrophedon decomposition [61]	49

Figure 23 - Map explored using LRV shown in (a), tree representation of map shown in (b) [62].....	50
Figure 24 - Indoor experimental setup for perimeter detection [64]	51
Figure 25 - Three robots defining a perimeter [64]	52
Figure 26 - Plot of the equation in Table 7 with an absolute difference function [65].....	53
Figure 27 - Cooperative exploration example [67].....	55
Figure 28 - Cooperative rendezvous example [68].....	55
Figure 29 - Comparison of recall-precision for SURF, SIFT, and GLOH on eight images from Mikolajczyk's database [25]	57
Figure 30 - Auto-correlation principal curvature space-heavy lines five corner/edge/flat classifications; fine lines are equi-response contours [23]......	60
Figure 31 - Sample screenshot of stereo feature tracking over time using Harris [42].....	61
Figure 32 - Example of dense stereo with original left and right image above [22].	68
Figure 33 - Screenshot of using SURF for optic flow	69
Figure 34 - Sample screenshot of LK-SURF tracking features; older frame on top, recent frame with tracked features on bottom	71
Figure 35 - Comparison of SURF vs. LK-SURF egomotion. Left: Path based on SURF egomotion. Right: Path based on LK-SURF egomotion	72
Figure 36 - Example of bland feature tracking. Top: features plotted as a distance away from the camera, stray point circled in red. Bottom: Original image with matched features, excess features tracked in the middle	74
Figure 37 - Comparison of EKF and UKF [27].....	76
Figure 38 - Comparison of Gamma and Gaussian distribution	78
Figure 39 - Sample Gaussian mixture.....	78
Figure 40 - Naive Multi-Gaussian recursive filter	81
Figure 41 - Consolidated approach to Multi-Gaussian recursive filtering	82
Figure 42 - Example sensor noise sampled over time	84
Figure 43 - Kalman estimate and sensor input data.....	84
Figure 44 - CUMRF estimate and sensor input data.....	84
Figure 45 - Comparison of Kalman and CUMRF estimate error	85

Figure 46 - Comparison of Robust Kalman Filter vs. Extended Kalman Filter. Left: Robust Kalman Filter path; Right: Extended Kalman Filter path; Bottom: Estimate of true path.....	96
Figure 47 - Comparison of EKF SLAM computation cost vs. Forgetful SLAM computation cost.....	109
Figure 48 - Comparison of Forgetful SLAM path to EKF-SLAM path. Left: estimate of true path; Center: Forgetful SLAM path; Right: EKF-SLAM path.....	110
Figure 49 - Overview of HAR-SLAM. Forgetful SLAM operating on current information. Links are established between past poses and landmarks.	114
Figure 50 - Diagram of Cooperative LP-SLAM.....	144
Figure 51 - Figure-eight path; Left: Forgetful SLAM path; Right: HAR-SLAM updated path.....	153
Figure 52 - HAR-SLAM map with corrected path and landmark locations.....	154
Figure 53 - Sonar produced from HAR-SLAM path; Left: Raw sonar values; Right: Filtered sonar values.....	155
Figure 54 - Hybrid sonar and visual landmark map on figure-eight path.....	155
Figure 55 - Hybrid sonar and visual landmark map on short counterclockwise loop (left) and long clockwise loop (right)	156
Figure 56 - CAD drawing of the test location; Left: layout with approximate location of furniture; Right: HAR-SLAM map and path drawn on top of the CAD drawing.....	157
Figure 57 - Paths with marked features for promotion; Top-Left: Figure-eight path; Top-Right: Long clockwise path; Bottom: Short counterclockwise loop	158
Figure 58 - Histogram of landmark strength, calculated by the landmark promotion metric for the figure-eight path	158
Figure 59 - Visual landmark maps generated by Cooperative LP-SLAM using Figure-eight path, long clockwise path, and short counterclockwise path.....	159
Figure 60 - Sonar maps generated by Cooperative LP-SLAM using Figure-eight path, long clockwise path, and short counterclockwise path.....	159
Figure 61 - View of the test platform.....	161
Figure 62 - CAD design of custom chassis modifications [83].....	162
Figure 63 - Diagram, schematic, and view of the sensor acquisition board [83].....	163

Figure 64 - Encoder parts and assembled encoder [83].....	164
Figure 65 - Close view of the TRX INU with Point Grey cameras, the Optical INU.....	165
Figure 66 - Freifunk OLSR mesh network visualizations [86] [87].....	167
Figure 67 - ASL Framework hierarchy.....	168
Figure 68 - ASL Framework data flow.....	169
Figure 69 - Sample screenshots of the ASL Application.....	170
Figure 70 - ASL network organization [87]	174
Figure 71 - Diagram of connecting devices (blue) and functional units (black) for Cooperative LP-SLAM.....	175
Figure 72 - Point Grey camera status and settings panels	177
Figure 73 - Parallax servo controller device status and parameter panel	179
Figure 74 - Xbox controller device status panel	179
Figure 75 - Mesh networking device status and parameter panels [87]	180
Figure 76 - Snapshot of stereo synchronizer functional unit	182
Figure 77 - Snapshot of stereo rectifier functional unit	182
Figure 78 - Snapshot of LK-SURF functional unit.....	183
Figure 79 - Encoder calibration environment [83]	188
Figure 80 - Histogram and Gaussian fit for sample encoder calibration	189
Figure 81 - Matlab toolbox calibration - camera-centered (left) and world- centered (right) views given multiple checkerboard samples.....	190
Figure 82 - Screenshot of stereo camera calibration in ASL.....	191
Figure 83 - Top: Sample unprocessed stereo image pair; Bottom: Rectified stereo image pair.....	192
Figure 84 - Diagram of an epipolar line [15].....	193
Figure 85 - Histogram of epipolar-based errors with a Gaussian Mixture Model fit.....	194
Figure 86 - Stereo camera projection distance mean errors and variances. Left: Scatter of mean error vs. distance. Right: Scatter of variance vs. distance.	196
Figure 87 - PI implementation of orientation estimator [9][15][85][84].....	197
Figure 88 - Histogram of stationary orientation data in one dimension. Left: Gaussian Model fit, failed KS-Test, Right: Gaussian Mixture Model fit, passed KS-Test.....	198
Figure 89 - Stereo view of gravity aligned checkerboard.....	199

Figure 90 - Inertial (blue) vs. optical (green) directions of gravity on unit sphere	199
Figure 91 - Multiple views of calibrated inertial (blue) and optical (green) directions of gravity on unit sphere	201
Figure 92 - Left: Four-wheeled vehicle with parallel steering [96]; Top-Right: Kinematic model [97]; Bottom-Right: Parallel steering maneuver [97]	201
Figure 93 - Diagram of three-layered neural net to describe robot model.....	203
Figure 94 - Screenshot of Matlab Neural Network Toolbox	204
Figure 95 - Test path of figure-eight around lab, units in meters. Black path is the best estimate of the actual path taken. Blue path is generated only by linear model predictions. Red path is generated only by nonlinear model predictions.	205
Figure 96 - Test path of four loops, units in meters. Black path is the best estimate of the actual path taken. Blue path is generated from linear model predictions. Red path is generated from nonlinear model predictions.....	206
Figure 97 - Architecture of a GeForce 8 GPU [99]	219
Figure 98 - Figure-eight pedestrian path; Left: Robust Kalman Filter with no model; Right: Path generated by TRX software	221
Figure 99 - Motion capture marker on Optical INU. Left: Normal view of the markers; Right: Lit version of the markers, showing the marker reflectivity.....	222
Figure 100 - Left: Motion capture camera from Natural Point; Right: Sample motion capture positioning and orientation estimate using rigid markers	222

CHAPTER 1: INTRODUCTION

1.1. Introduction to SLAM

Simultaneous Localization and Mapping (SLAM) is a topic of high interest in the robotics community. The ability for a robotic system to map an unknown environment while locating itself and others within the environment is of great use in robotics and navigation in general. It is especially helpful in areas where GPS technology cannot function, such as indoor environments, urban canyons, and caves. The idea of SLAM was introduced at the 1986 IEEE Robotics and Automation Conference in San Francisco, California, where researchers Peter Cheeseman, Jim Crowley, and Hugh Durrant-Whyte began looking at robotics and applying estimation-theoretic methods to mapping and localization [1]. Several others aided in the creation of SLAM, such as Raja Chatila, Oliver Faugeras, and Randal Smith, among others [1].

At a theoretical and conceptual level, SLAM is considered a solved problem. Many successful implementations, using sonar, laser ranger-finders, or visual sensors, exist [2] [3]. SLAM has been formulated and solved theoretically in various forms. However, substantial practical issues remain in realizing more general SLAM solutions, notably those that relate to constructing and using perceptually rich maps as part of a SLAM algorithm [1]. For large-scale and complex environments, especially those requiring full 3D mapping, the SLAM problem is still an open research issue [4]. Most SLAM solutions work in small areas, on a 2D surface with 2D landmarks. The

introduction of a third dimension in either landmark state or robot state complicates traditional SLAM solutions [4].

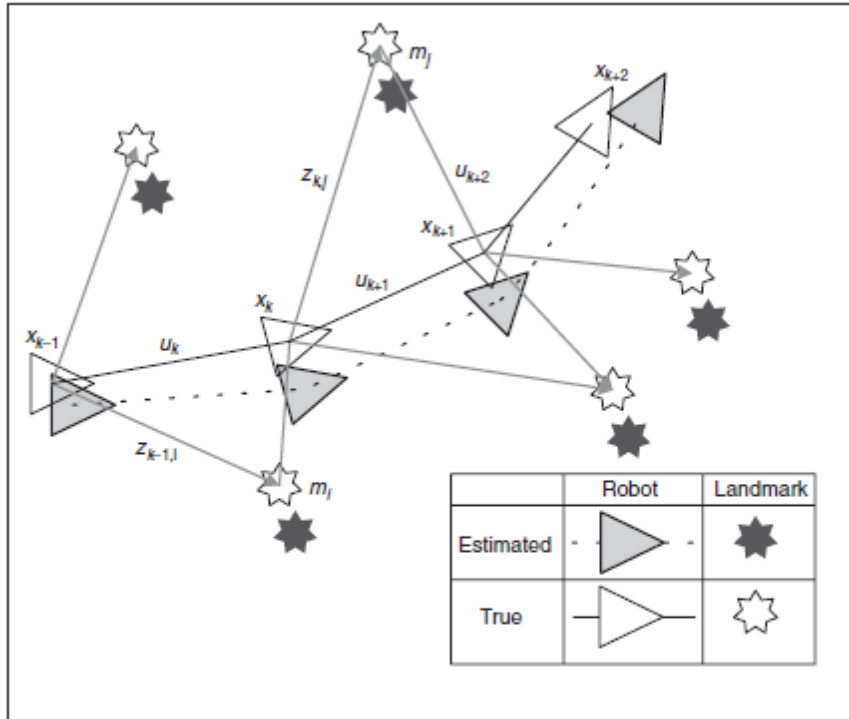


Figure 1 - The essential SLAM problem [1]

Generally, SLAM techniques are driven by exteroceptive sensors such as cameras, laser range finders, sonars, and proprioceptive sensors (for example, wheel encoders, gyroscopes, and accelerometers) [5]. Figure 1 shows the basics of the SLAM problem: a robot travels in an unknown environment, estimating its own position and its position relative to several landmarks in the area. It maps its position and the landmarks around it in successive iterations based on new observations. The problem of SLAM can be defined using the theory of uncertain geometry [6]. There are two important elements in uncertain geometry: the actual geometry of structures, and the overall topology of relations between geometric structures. With uncertain geometry, the representation by a parameterized function and a probability density of all geometric locations and features

provides a simple means of describing geometric uncertainty. Using this representation, the analysis of uncertain geometry can be reduced to a problem in the transformation and manipulation of random variables [6]. The key elements of uncertain geometry theory are providing a homogeneous description of uncertain geometric features, developing procedures to transform and manipulate these descriptions, and supplying a mechanism for the consistent interpretation of relations between uncertain features [6].

According to [7], navigation problems can be categorized into several sub-problems. First, there is the distinction between indoor and outdoor navigation. Indoor navigation is composed of three sub-types: map-based navigation, map-building navigation, and map-less navigation. Map-based navigation systems depend on user-created geometric models or topological maps of the environment. Map-building-based navigation systems use sensors to construct their own geometric or topological models of the environment and then use their models for navigation. Map-less navigation systems use no explicit representation of the space in which navigation is to take place. Rather, they rely on recognizing objects found in their environment. Outdoor navigation can be divided into two classes according to the structural level of the environment. There are structured environments, such as road systems and urban landscapes, and there are unstructured environments, such as terrain and planet surfaces. For the purposes of this dissertation, only map-building-based navigation, which is the core of SLAM, and map-less navigation, are considered. Both are considered solely in indoor environments.

According to [8], large-scale SLAM, which consists of maps with more than a few thousand landmarks or maps that traverse over distances greater than a hundred meters poses three main challenges. The first is algorithmic complexity, which is

inherent in any application that handles thousands of pieces of information. The second is a consistency issue, whereby we have estimation techniques inhibited and weakened by large distances, the consideration of 3D motions, and sensor noise in general. The third difficulty is the loop-closing detection and feature matching issue. Loop-closing involved recognizing that the robot has returned to a mapped area, and propagating corrections to the entire map correcting any inconsistency. Loop-closing and feature matching in general are difficult to solve using landmark and robot position estimates alone, since they tend to be inconsistent over long distances. To overcome these problems, this dissertation will cover the development of several techniques that will be incorporated into a final cooperative SLAM solution.

1.2. Problem Definition

The purpose of this dissertation is to create computationally efficient and robust algorithms that will allow a group of robots to perform cooperative SLAM. A common coordinate system of the entire region (building) will be created such that the sensor platforms and any anomalies or landmarks will have coordinates that are relevant to the robot group. The goal is to fuse various forms of sensor readings, including stereo vision readings, odometry readings, and inertial measurements to achieve a more accurate map than is possible by any one sensor. The goal of this dissertation is to develop a set of inexpensive test platforms, and a set of efficient SLAM algorithms that can carry out experiments supporting the advances proposed in this dissertation.

1.2.1. Purpose

The focus of this research is to apply advanced algorithms to simple hardware platforms in order to map regions in GPS-denied environments. Many robotic platforms used for this purpose employ expensive sensors and focus only on performance and not the implementation. A LIDAR sensor, for instance, could cost \$5,000. A goal of this dissertation is to create an entire robot that is capable of cooperative mapping for less than the cost of a single LIDAR sensor (under \$5,000). Cheap, off-the-shelf components will be used so that the robots are easy to build and expendable in certain missions. Using the robots to construct maps in GPS-denied areas will provide platforms a common map and coordinate system that can be used for other missions such as search and rescue, bomb disposal tasks, and others.

1.2.2. Challenges

Building a set of low cost robots and the associated data-sharing network has been an ongoing effort over several years in the Autonomous Systems Lab (ASL). The key challenges to overcome in the development and implementation of cooperative mapping SLAM are managing sensor inaccuracy, map consistency, and computational complexity. Inertial measurement units, such as the one used in our experiments, have inherent noise. To perform dead reckoning positioning based on the accelerometer data, requires double integration of the data. Visual odometry requires single integration of data; however, it is difficult to find enough visual features to determine the motion of a robot. In addition, it is difficult to track points in a sequence of images across varying lighting conditions. Integrating motion information, from inertial or visual data, results in accumulating errors

that are added to the path and map. Integrating errors causes position estimates to drift away from the true value, which makes the SLAM problem very important. With SLAM, building a map and tracking landmarks over time is computationally and spatially expensive. Both the computational expense and storage expense grows approximately with the number of landmarks squared.

Combining stereo vision with inertial navigation also proves challenging. In order to combine the two systems, high-speed computation is required to process the real-time images such that they match inertial measurements and correct them. False points and poorly tracked features will have to be removed from the visual data, and a considerable amount of calculations will go into combining the two systems. However, the hybrid system will drastically improve the dead reckoning of the system.

Image processing will also be difficult. Finding features to track, as near to real-time as possible for ego-motion calculations will be challenging. Additionally, 3D visual landmarks need to be investigated, characterized, and encoded such that they are distinct from other landmarks yet flexible enough to be recognized from various approaches.

Creating a cooperative SLAM system across many peer-to-peer nodes also poses a large challenge. With cooperative SLAM, there must be a networking system in place to accommodate a vast amount of real-time data sharing to facilitate mapping. The robots will have to share data with one another in order for them to build a consistent map. Algorithms will have to function without a lead or master robot, because every unit will be a peer. Therefore, no one robot can instruct other robots to perform a given action.

1.3. Hardware Selection

To create cost effective robots, inexpensive and data-rich sensors were selected. A micro electromechanical system (MEMS) inertial navigation unit (INU) from TRX Systems was selected for gyroscopic readings. In current MEMS INUs, the accelerometers are only good enough to indicate the direction of gravity. Inexpensive machine vision cameras were selected for stereo vision, giving a rich set of ranges and landmark identification. Small pinging ultrasonic ranging sensors were used for obstacle avoidance and wall detection. An off-the-shelf remote control (RC) truck was used for the chassis. Most components were purchased off-the-shelf and assembled to construct the robots. The computing platform is an AOpen Mini PC, and a standard wireless router is used for networking. A DC voltage adapter and servo controller were purchased to help connect all these pieces. Additionally, an acquisition board was custom designed using a Microchip dsPIC microcontroller. Each robot costs approximately \$2000 - \$3000 (significantly beating our target of \$5000), with the RC truck, INU, and computer being the three most expensive items.

1.4. Solution

The final solution to cooperative SLAM required integration of a series of theoretical advances and implementation advances. New theoretical advances include a nonlinear and non-Gaussian recursive (Kalman-like) Filter, a Robust Kalman Filter, and a tiered architecture for SLAM. New implementation advances include an image-based

feature tracker, accurate camera calibration, an optical system with an inertial sensor calibration, encoder calibration, and neural net system modeling.

A new image feature tracker was created to perform stereo correspondence, identify features, and reliably track them over time. This new method was called LK-SURF, since it leverages the Lucas-Kanade Tracker, and Speeded-Up Robust Features (SURF). This allowed each robot to use the inexpensive cameras to range to features accurately and establish a known correspondence, which simplifies the SLAM problem.

Several developments were made in filtering algorithms to make a new temporary SLAM called Forgetful SLAM. This method combines a kind of “Robust Kalman Filter,” and nonlinear and non-Gaussian recursive filters to filter both robot positions, and the locations of visual features. This method can be extended to other sets of sensor observations that can be tracked. As its name implies, Forgetful SLAM does not retain features, but refines observations, refines the robot state, and correlates observed data.

Hierarchical Active Ripple SLAM (HAR-SLAM) was created to handle maps with a very large number of features, turn maps into chains of Kalman Updates, and handle multiple robot mapping. Landmark Promotion SLAM (LP-SLAM) uses HAR-SLAM as a theoretical backend, but limits what information is passed up the various levels of HAR-SLAM, making the system more robust and efficient. LP-SLAM is an implementation algorithm while HAR-SLAM is a theoretical contribution. LP-SLAM covers landmark spatial storage and landmark matching.

The result of this dissertation is a set of theoretical and implementation contributions to cooperative SLAM. Contributions include improved image processing

techniques, novel calibration methods, more efficient SLAM techniques, and multi-robot SLAM techniques.

CHAPTER 2: LITERATURE REVIEW

This chapter contains a literature review of four general topics that are relevant to this dissertation. The first involves fusing inertial and visual sensors. This area of research is quite useful as cameras and inertial sensors are inexpensive ways of extending current dead reckoning and simultaneously tracking the environment. The second topic involves SLAM in general, covering many sub-topics, including Kalman Filters, Particle Filters, and various types of SLAM that use different sensing techniques. The third topic involves cooperative control and map coverage, specifically how other researchers in the robotics and SLAM fields have handled cooperative control. The fourth topic involves landmark acquisition, visual descriptors, and feature tracking. These four topics give a general background for the advances presented in this dissertation.

2.1. Fusing Inertial and Visual Sensors

There is an inherent problem with using any one sensor, and that is the accumulation of error. Inertial measurement, which includes the use of accelerometers and gyroscopes, is a prime example. Positioning using accelerometers requires the integration of sensor information twice after filtering, and the orientation requires a single integration. The noise from these sensors accumulates over time and results in incorrect position and orientation. Inertial measurement is very susceptible to drift, especially when stationary [9]. Vision sensors pose an almost opposite concern. When motion is

slow and controlled, visual odometry is very accurate and does not introduce errors nearly as much as inertial systems do. However, visual sensors are limited when there are no visual features to track, or when the motion is too fast to track features. Similar to dead reckoning, visual odometry accumulates error over time. Nevertheless, it has been shown that visual odometry provides more accurate results than most sensor combinations when compared to dead reckoning [10]. Fusing the two systems is a natural solution for improving a positioning system.

Early SLAM approaches did not use vision as a means of odometry. As a result, they were unable to build consistent maps over long periods due to accumulating sensor errors or physical disturbances [11]. Combining visual sensors with inertial sensors will lengthen the period in which a human or vehicle can navigate in GPS-deprived environments [12]. Outdoor navigation systems usually utilize GPS as a means of correcting drift, but indoors there is no global coordinate system.

2.1.1. Visual Odometry

Visual odometry is known as ego-motion. Essentially, it involves the calculation of relative camera pose over sequential images. Unfortunately, visual odometry is computationally quite complex. Due to performance limitations, a real-time navigation system can typically utilize only a small number of landmarks for computing the camera pose [13]. Many techniques reduce the number of features used in visual odometry. In [13], the authors imposed constraints on the distance between features, on orthogonal planes. However, these simple heuristics do not necessarily remove false points.

Several researchers have implemented visual odometry. Examples would be in [8] and [10], where the authors used Harris Corners as features in a stereo vision system. They made motion predictions using visual motion estimation, which used matched 3D points from two time frames to calculate the 6 DoF movement. A 3D transformation was determined by that minimizes the square of the error between the points in the first time frame and the transformed matched points in the second time frame, that is least-squares minimization. The 3D transformation matrix T that minimizes the mean-squares objective function is $F(T) = \frac{1}{N} \sum_{i=1}^N \|TP_2^i - P_1^i\|^2$. Least-squares rotation and translation were computed using the dual number quaternion method [10].

Another example of visual odometry was implemented by [14]. The authors presented a method that estimated a robot's displacement in an unstructured outdoor environment. The method used 3D point sets, which were produced using stereo vision frames. Points were tracked between frames such that proper associations were made. The algorithm was able to estimate the six parameters of the robot position with a relative error smaller than about 5% after processing hundreds of images over tens of meters. Pixel selection was performed in three steps. The first was an a priori selection done based on selected matching features in the stereo images. Second, an empirical model of the pixel-tracking algorithm was used to remove falsely tracked pixels. Finally, an outlier rejection was performed when pixels did not match the computed estimate of displacement between two stereo sets. The main localization loop lasted about 7-8 seconds (image acquisition, stereo-correlation and motion estimation). Though the algorithm worked well, it was too computationally complex for real-time performance.

2.1.2. Inertial Navigation

Inertial navigation uses accelerometer and gyroscope measurements to compute position and orientation offsets from a starting position and orientation. Simply integrating the sensor data once it has been filtered is an easy and computationally simple approach. However, this method is very susceptible to sensor drift that can grow quickly in MEMS sensors [15]. Inertial navigation units (INUs) are plagued with random walk, a phenomenon in which a stationary INU yields outputs indicative of motion. This is caused by integrating the noise inherent in the sensor integration. Sometimes, the noise can be large enough that humans and robots being tracked by INUs are offset by the size of an entire building, meaning that errors can accumulate to drastically large values [15]. Figure 2 shows an example of the effects of drift on mapping hallways that are at right angles [16].

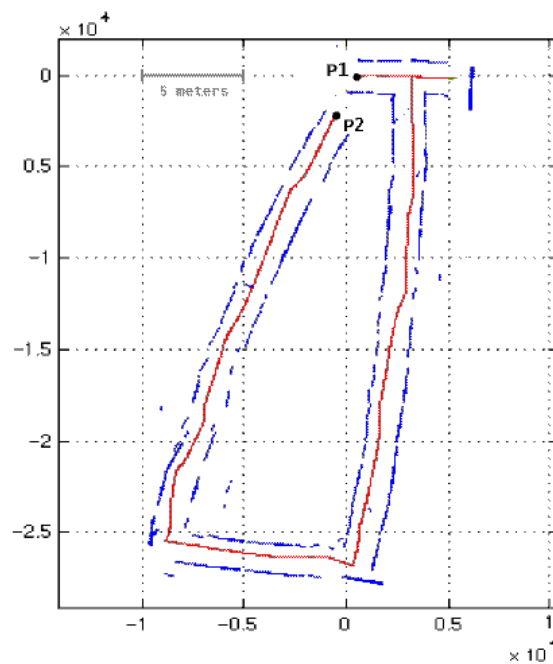


Figure 2 - The effect of gyroscope drift on mapping [16]

2.1.3. Inspiration from Biology

Current SLAM systems use dead reckoning techniques, such as inertial measurements or odometry as the basis for navigation. However, biology suggests a different approach. Humans use their visual surroundings to correct their own inertial sensors. When the two systems are discordant, humans experience disorienting illusions that depend on the room's relative position to gravity and to themselves. In humans, vision, and visual landmarks in particular, undoubtedly play a dominant role in spatial orientation [17]. An interesting fact about the human vision system is that, during normal eye movements, the world is not perceived as moving [18]. According to [18], the author suggests that the textural background image, whatever its relation to the anatomical retina, always tends to determine the phenomenal environment. The more it approximates the total image, the greater the stability. However, humans do not rely solely on vision for navigation. For instance, in environments devoid of light, or during quick turns, visual systems are unreliable. In darkness, human navigation relies mostly on vestibular senses, which are similar to inertial sensors. Human navigation also relies on proprioceptive inflow and motor outflow from the locomotive system, which is similar to motion prediction and wheel encoders [17]. In robotics, we use wheel encoders to count the number of times a tire rotates, which is like counting the number of steps a person takes.

Biological systems naturally combine internally sensed motion with vision. Understanding more about how accurate these internal sensors are may give some insight to what is possible. One paper in particular [17] utilizes various experiments and conditions to determine the accuracy of human inertial sensing. The researchers

conducted all experiments in complete darkness and deafened the subjects with white noise to remove any possibilities of visual or auditory cues. All experiments required a human participant to stand upon a rotatable platform and make estimations of how much the platform had rotated. There were four conditions tested in this experiment: condition PE (passive rotation, verbal estimation upon request), condition AE (active turning, verbal estimation upon request), condition PT (passive rotation, targeting), and condition AT (active turning, targeting). Three rotational speeds were tested: slow ($18.5^\circ/\text{s}$), medium ($37^\circ/\text{s}$), and fast ($55^\circ/\text{s}$). Condition PE involved spinning the platform at set speeds and, when a beeper went off, subjects guessed the amount of rotation that had occurred. Condition AE required the subjects to spin on the platform at set speeds and then verbally indicated how much they thought they had turned. Condition PT let the subjects control the platform and spin it to a certain displacement that they were instructed to achieve. Condition AT let the subjects spin on the platform until they felt they were displaced the required amount.

The results from [17] are shown in Figure 3 and Table 1. In Figure 3, the results of all experiments are plotted with the y-axis as the average human-estimated speed and the x-axis as the actual displacement. The plot shows a large discrepancy around the speeds and a drift in the human estimations. Table 1 shows the averages per condition and speed. The results indicate that human estimation of speed is far from accurate, and the high standard deviation indicates that human estimation is not consistent either. The results indicate that human inertial sensors are not accurate on their own and drift drastically. When the system is compensated with motor movement and measurement, a more accurate result is shown with less drift.

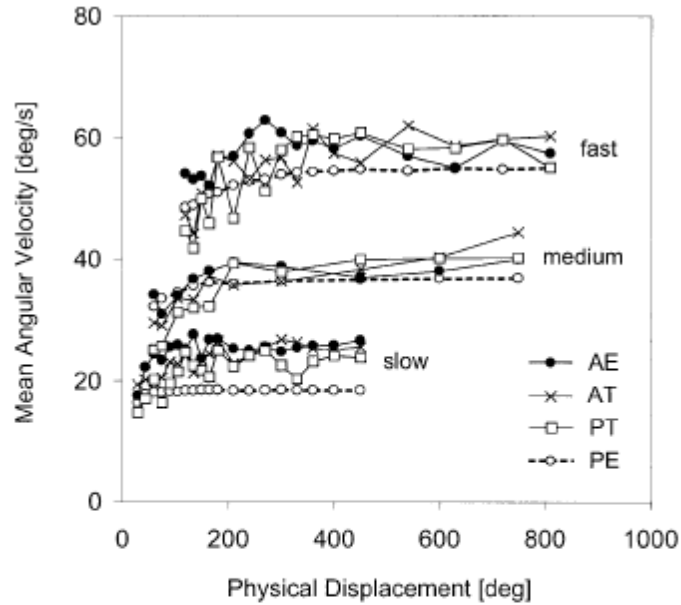


Figure 3 - Mean angular velocity vs. physical displacement [17]

		PE	PT	AE	AT
Slow (18.5°/s)	Average	18.2	21.5	24.7	23.5
	Std. Dev.	0.2	4.1	6.0	5.4
Medium (37°/s)	Average	53.4	34.3	36.7	35.7
	Std. Dev.	0.5	5.3	6.9	6.5
Fast (55°/s)	Average	52.8	55.0	57.5	55.0
	Std. Dev.	0.6	6.8	10.5	7.2

Table 1 - Averages and standard deviations for various conditions and speeds [17]

Understanding the effect vision has on inertial perception in biological systems can give insights on how sensor fusion is done. In one paper [19], the authors test and summarize various orientation experiments involving humans in moving rooms. Most experiments had a luminous square frame in dark surroundings. A frame tilted 28° was found to induce a mean illusory tilt of subjects' bodies of 2.8° . Tilting the frame 28° in the frontal plane caused a vertical rod to appear tilted by 6° on average in the opposite direction, but with large variability between subjects. For a room inclination of 30° , the mean rod setting of five subjects was 13.2° in the direction of room pitch. Upright

subjects exposed to a large textured display rotating about the roll body axis felt as if they were moving continuously in the opposite direction. Interestingly, they felt tilted at a mean constant angle of 15° . Figure 4 is a diagram depicting the various frames of reference that existed when the room or frame was tilted. The interesting conclusion revealed by these experiments is that a person's vision plays a key role in his perceived spatial orientation. The findings indicate that humans combine their visual and inertial input to predict their position and orientation. Fusion of visual and inertial systems in robots would mimic this key biological technique.

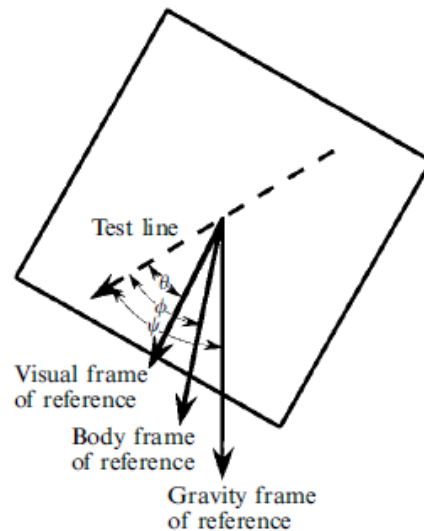


Figure 4 - Three frames of reference for perceived orientation of a test line with respect to gravity [19]

2.1.4. Examples of Fused Data

Fusion of visual and inertial sensors has been performed by a few researchers. One of the earliest and most noteworthy approaches was done in [20]. The authors describe how stereo vision was fused with inertial information in order to recover 3D segments. This initial system was limited in the sense that it only used accelerometer data and only used vertical line segments for stereo features. However, the results are

still relevant to this discussion. Figure 5 shows a diagram of how vertical lines were used for stereo correspondence. Figure 6 shows a pair of sample stereo images with detected vertical lines and detected edges selected. The ends of the vertical lines were used as a single 3D point in calculating camera pose. The paper did not yield superb results, but it introduced the novel concept of fusing visual odometry and inertial navigation.

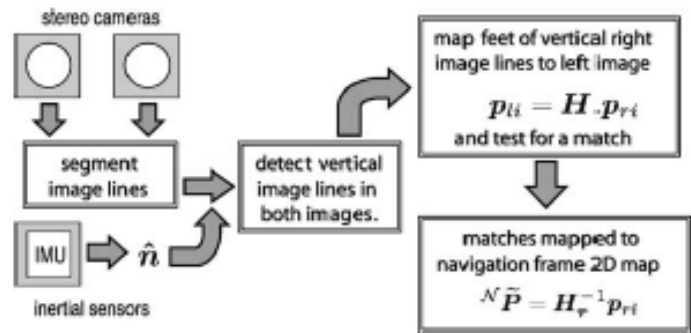


Figure 5 - 3D reconstruction algorithm [20]

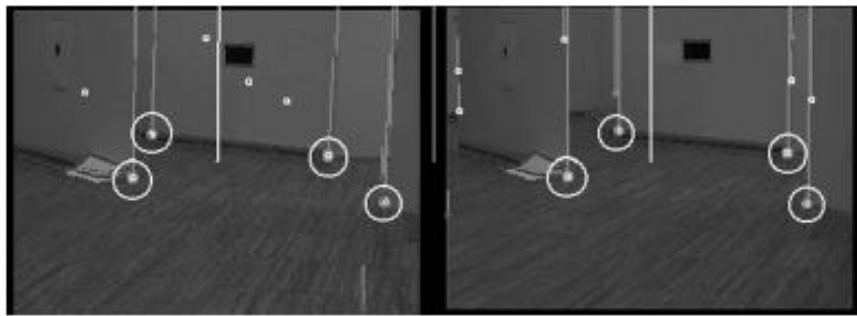


Figure 6 - Stereo image with detected edges and vertical lines [20]

After the work presented in [20], other researchers attempted to fuse visual odometry and inertial measurement. Another, yet ineffectual, approach to fusing visual odometry and inertial measurement used the information from a single camera to derive the odometry in the plane and fuse it with roll and pitch information derived from an on-board inertial measurement unit (IMU) to extend to three-dimensions, thus providing

odometric altitude as well as traditional x and y translation [21]. This approach used OpenCV's [22] implementation of Harris Corners [23] with pyramidal Lukas-Kanade optical flow calculation [24] of the concrete and asphalt surfaces. However, the technique proved to be very poor. Other feature trackers such as the Canny edge detector [22] and Speeded-Up Robust Features (SURF) [25] were also implemented without any significant improvements. The approach that worked best was correlation matching, also known as template matching. Template matching involves taking a patch of an image, or the entire image, and finding the sum of the absolute differences. By meeting a threshold, using the sum of absolute differences, results in a successful match. Template matching proved to be much more reliable and produced precise displacements on all the surfaces tested: concrete, asphalt, gravel, and grass [21].

Other researchers [26] used a modified linear Kalman filter to perform the data fusion, because it reduced drift over time. The Kalman filter lengthened the period during which a human or vehicle could navigate in GPS-deprived environments. The state vector contained only inertial sensor errors related to position but no landmark information was tracked, thus no SLAM algorithms were performed. Only tracking inertial sensor error allowed uncertainty to be properly represented by a covariance matrix. Image data contributed stochastic epipolar constraints over a period of time and space, which improved observability of the IMU error states. The method tracked sparse independent corner features in order to avoid the texture ambiguity associated with optical flow, to reduce the accumulation of inertial drift over time, to facilitate detection of visual distractions, and to bound computational requirements. This paper relied on the gyroscopes to compensate for camera rotation, so image data was not used to improve the

gyroscopes, only the translation of the robot. Figure 7 shows a single feature tracked over multiple frames. Any two camera poses and a jointly observed feature define a plane. The paper did yield impressive results, but the important notion was the use of visual feedback as constraints on inertial measurements. In one test scenario, the camera moved through an arbitrary closed-loop path, carried by a person. The unaided IMU drifted by 262 meters during the run. With visual correction, the drift was reduced to 89 meters (66% improvement).

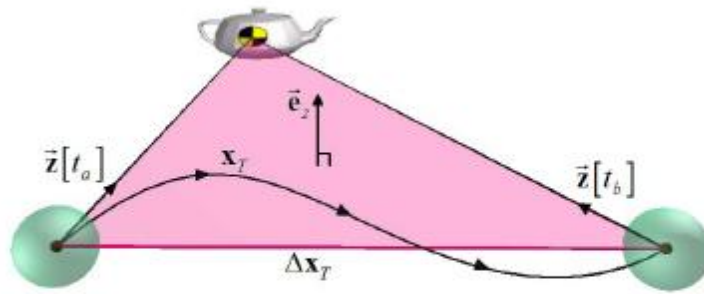


Figure 7 - The epipolar constraint, satisfied by ideal sensor data [26]

2.2. Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is a key concept in robotics. It allows robots to explore unknown environments and retain a stable coordinate system that can be used for path planning and cooperation. SLAM can even be used as a tracking method for humans and vehicles in GPS-denied environments where a global coordinate system fails. In essence, SLAM builds a relative coordinate system from local information. Figure 8 is a prime example of why SLAM is required for tracking position. The image is of a robot's path measured using odometry readings, which simply measure the amount each wheel turns. This method is inaccurate and drifts over time as wheels

slip or deform. Figure 8 is intended to show the odometric position of the robot as the robot travels through parallel paths. However, the drift from the sensor readings prevents an accurate trajectory from being calculated.

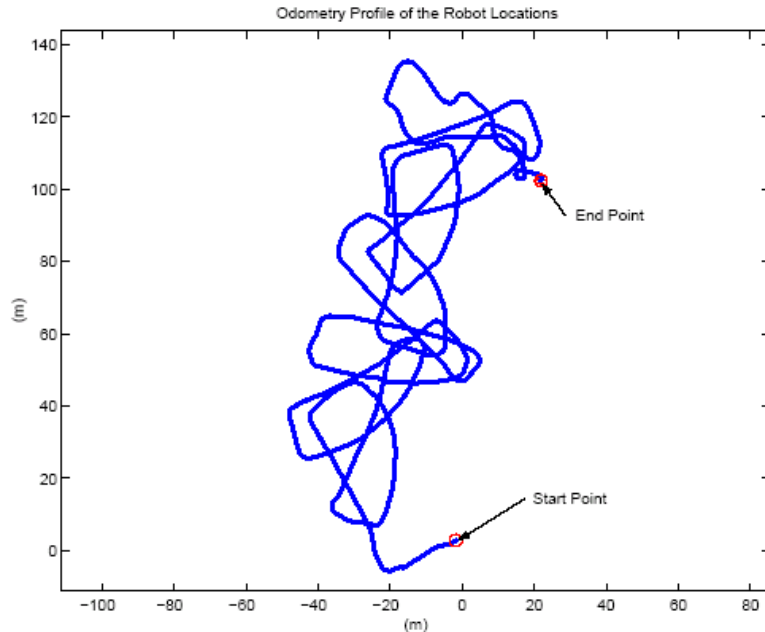


Figure 8 - Drift in odometry [27]

In general, there are three main approaches to solving SLAM: the estimation-theoretic (or Kalman Filter) approach, the qualitative approach, and the measures to describe uncertainty approach [28]. The most common approach to solving SLAM is the estimation-theoretic (or Kalman Filter) approach. This method provides a recursive solution to the navigation problem. In addition, it provides a means of computing estimates for the uncertainty in vehicle and map landmark locations based on statistical models for vehicle motion and relative landmark observations, which are all consistent with one another.

The second method is the qualitative approach. The qualitative approach filters observations using heuristics, and corrects the map and location using non-mathematical

means. Using this method for navigation and the general SLAM problem has advantages over the first approach, because it limits the need for accurate models. It requires less computation, and it uses an approach similar to that used by biological systems. For example, in buildings, a qualitative approach is to snap to hallways, or when driving, a qualitative approach is to ensure the car is on the road. Observations can be filtered by using facts about building size or robot size, and can help build the map.

The third approach involves the use of iconic landmark matching, global map registration, bounded regions, and other measures to describe uncertainty. Typically, this method uses a Bayesian approach to map building that does not assume Gaussian probability distributions, which are required by the Kalman filter. Examples of this style would be the Particle Filter, which uses brute force random combinations to determine probabilities. This technique, while very effective for localization with respect to maps, does not lend itself to incremental SLAM solutions wherein a map is gradually built as information is received from sensors, instead several brute force possibilities exist.

Regarding the use of the estimation-theoretic (or Kalman filter) approach, one research group [28] proved a few important points relevant to this discussion. Firstly, they proved that the determinant of any sub-matrix of the map covariance matrix decreases monotonically as observations are successively made. Secondly, they proved that, as the limit as the number of observations increases, the landmark estimates become fully correlated. The proofs in their paper [28] indicate that the SLAM problem's entire structure hinges on maintaining complete knowledge of the cross-correlation between landmark estimates. Figure 9 shows a diagram depicting the cross-correlation between landmarks. Minimizing or ignoring cross-correlations is contrary to the structure of the

problem, as critical information is lost. As the vehicle progresses through the environment, errors in the estimates of any pair of landmarks always become more correlated. Eventually, the errors in the estimates of any pair of landmarks become fully correlated. Therefore, given the exact location of any one landmark, the location of any other landmark in the map can also be determined with absolute certainty. As the vehicle moves through the environment, taking observations of individual landmarks, the error in relative location estimates between different landmarks reduces monotonically until the map of relative locations is known with absolute precision.

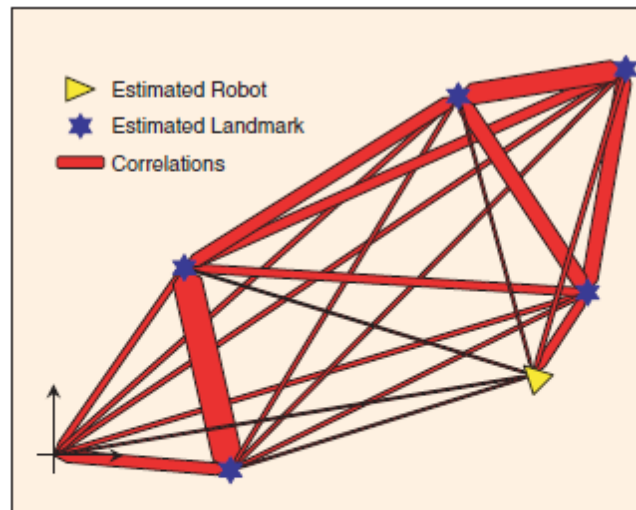


Figure 9 - SLAM model using a spring network analogy [1].

Regardless of the aforementioned SLAM technique, SLAM can use any distinctive-environment feature that is specific to a particular area for location-based navigation [29]. Such features can be as simple as a table or a sound, or as complex as a magnetic abnormality or a painting on the wall. The key pitfall to SLAM is that, as the number of landmarks increases, the computation required at each step increases as a square of the number of landmarks. Required map storage increases as a square of the number of landmarks as well [28]. The computational complexity associated with SLAM

can be reduced by a piecewise approach of subdividing the landmarks and associations into smaller maps, as well as by reducing the observability and covariance matrix of the problem by making it more sparse [30].

Despite all the aforementioned categories and approaches, most SLAM algorithms use one of two frameworks: the Extended Kalman Filter or the Rao-Blackwellized Particle Filter [31]. The next two main topics of this section will discuss Kalman filters and particle filters. Various offshoots of the Kalman filter will be discussed, as well as specific particle filters such as the Rao-Blackwellized Particle Filter. The remainder of this section will be devoted to specific types of SLAM and various implementations that have been tried.

2.2.1. *Kalman Filters*

The Kalman filter is a recursive estimator [32]. This means that only the previous state and the current measurement are needed to compute an estimate for the current state. Unlike batch estimation techniques, no history of observations or estimates is required. A Kalman filter is one of the few filters that was developed based on the time domain analysis. Most filters are formulated in the frequency domain and then transformed back into the time domain for implementation (for example, a low-pass filter). The Kalman filter offers many advantages over other estimation methods. Such advantages include temporal filtering, recursive implementation and being able to function with many types of observations [33].

2.2.1.1.

Basic Kalman Filter

The Basic Kalman Filter is the original Kalman Filter. This filter has several assumptions. With all filters presented in this dissertation, the system is assumed a Markov process. This is where the past and future data are independent, allowing recursive updates that do not include the entire past. Furthermore, the Basic Kalman Filter assumes the system is linear with Gaussian noise that is additive. The state of the filter is represented by two variables:

$\mathbf{x}_{k|k}$: the estimate of the state at time k given observations up to time k and

$\mathbf{P}_{k|k}$: the error covariance matrix (estimated accuracy of the state estimate).

The filter has two phases: predict and update. The predict phase uses the previous state estimate to produce an estimate of the current state. In the update phase, measurement information at the current time is used to refine this prediction to arrive at a new, more accurate state estimate for the current state [32]. This process does not always return accurate results, but given reasonable measurements that can be approximated as a linear function of the state with additive zero mean Gaussian noise and a good linear model with additive zero mean Gaussian noise that is a decent approximation of the system in the operating range, then it usually does perform well. Table 2 shows the Basic Kalman Filter equations. How each state is recursively calculated, and how predictions are made from the previous state, are shown.

System Equations:	$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$ $\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$ $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$ $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$
Prediction Phase:	
Predicted state:	$\mathbf{x}_{k k-1} = \mathbf{F}_k \mathbf{x}_{k-1 k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}$
Predicted estimate covariance:	$\mathbf{P}_{k k-1} = \mathbf{F}_k \mathbf{P}_{k-1 k-1} \mathbf{F}_k^T + \mathbf{Q}_{k-1}$
Update Phase:	
Measurement residual:	$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k \mathbf{x}_{k k-1}$
Residual covariance:	$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k k-1} \mathbf{H}_k^T + \mathbf{R}_k$
Kalman gain:	$\mathbf{K}_k = \mathbf{P}_{k k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$
Updated state estimate:	$\mathbf{x}_{k k} = \mathbf{x}_{k k-1} + \mathbf{K}_k \mathbf{y}_k$
Updated estimate covariance:	$\mathbf{P}_{k k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k k-1}$

Table 2 - Basic Kalman Filter equations [32]

The formulas in Table 2 contain the updated estimate covariance for the optimal Kalman gain. The system equations contain a state \mathbf{x}_k ; the state prediction involves the past state and current control. The state prediction has uncertainty noted as \mathbf{w}_k , which is modeled as a Gaussian with zero-mean and covariance \mathbf{Q}_k . The observation is noted as \mathbf{z}_k , with uncertainty \mathbf{v}_k . The observation uncertainty is modeled as zero-mean Gaussian with \mathbf{R}_k as the covariance. Other choices for the gain will have different estimators [32]. It is also worth noting that all estimates are zero mean. The state the probability distribution takes on a Markov type property, rendering all past information irrelevant compared to the most recent information. Figure 10 shows a linear propagation of Gaussian error, which is how the Basic Kalman Filter operates.

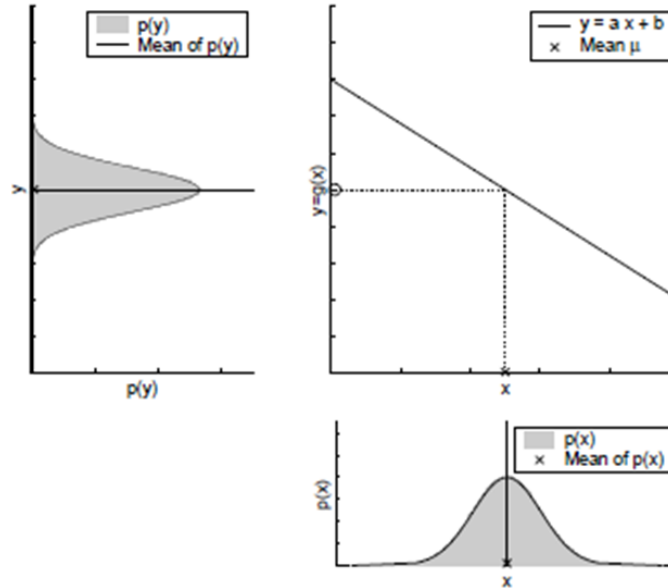


Figure 10 - Linear transformation of probability [27]

2.2.1.2. Extended Kalman Filter

The Extended Kalman Filter (EKF) is very popular with SLAM researchers [34]. Several research groups have based their approaches on an EKF: [4][8][33][35][36][37][38][39][40][41][42][43]. The EKF is a nonlinear version of the Kalman Filter. The F, B, and H matrices in the Kalman Filter begin as nonlinear functions, but in the prediction estimates, these functions are linearized. Table 3 shows the basic formulation of the EKF. The formulation is quite similar to the Basic Kalman Filter, except for the nonlinear system equations and the linearization. For a SLAM application, the robot state distribution is assumed to be independent of the landmarks' state distributions and both are assumed to be independent and follow Gaussian distributions [44].

System Equations:	$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$ $\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$ $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$ $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$
Prediction Phase:	
Predicted state:	$\mathbf{x}_{k k-1} = f(\mathbf{x}_{k-1 k-1}, \mathbf{u}_k)$
Predicted estimate covariance:	$\mathbf{P}_{k k-1} = \mathbf{F}_k \mathbf{P}_{k-1 k-1} \mathbf{F}_k^T + \mathbf{Q}_k$
Update Phase:	
Measurement residual:	$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}_{k k-1})$
Residual covariance:	$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k k-1} \mathbf{H}_k^T + \mathbf{R}_k$
Kalman gain:	$\mathbf{K}_k = \mathbf{P}_{k k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$
Updated state estimate:	$\mathbf{x}_{k k} = \mathbf{x}_{k k-1} + \mathbf{K}_k \mathbf{y}_k$
Updated estimate covariance:	$\mathbf{P}_{k k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k k-1}$
Jacobian Linearization:	$\mathbf{F}_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right _{\mathbf{x}_{k-1 k-1}, \mathbf{u}_k}$ $\mathbf{H}_k = \left. \frac{\partial h}{\partial \mathbf{x}} \right _{\mathbf{x}_{k k-1}}$

Table 3 - Extended Kalman Filter equations [45]

However, the EKF has many shortcomings. Generally, it is not an optimal estimator for nonlinear systems [45]. Due to its linearization, the filter may quickly diverge if the initial estimate of the state is wrong or if the process is modeled incorrectly [45]. Figure 11 illustrates how the EKF propagates the mean and covariance after linearizing the system. Figure 12 shows how a linearized version not only incorrectly estimates the true noise, but also incorrectly estimates the mean and variance of the true data. Another problem with the EKF is that the estimated covariance matrix tends to underestimate the true covariance matrix and risks becoming statistically inconsistent [45]. Because of linearization, the EKF suffers from computational complexity and

inaccuracy [46]. In addition, the EKF covariance matrices are quadratic in the size of the map, and updating them requires time quadratic in the number of landmarks [1][4][38][47]. In experiments, the Kalman Filter update time begins to grow rapidly when the number of features approaches 100 [38].

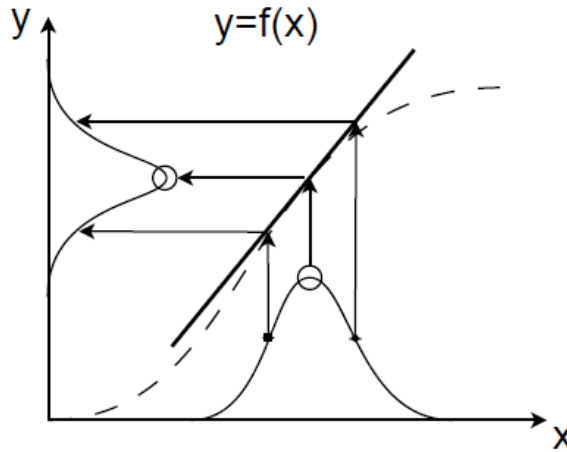


Figure 11 - Example of linearization for Gaussian propagation [45]

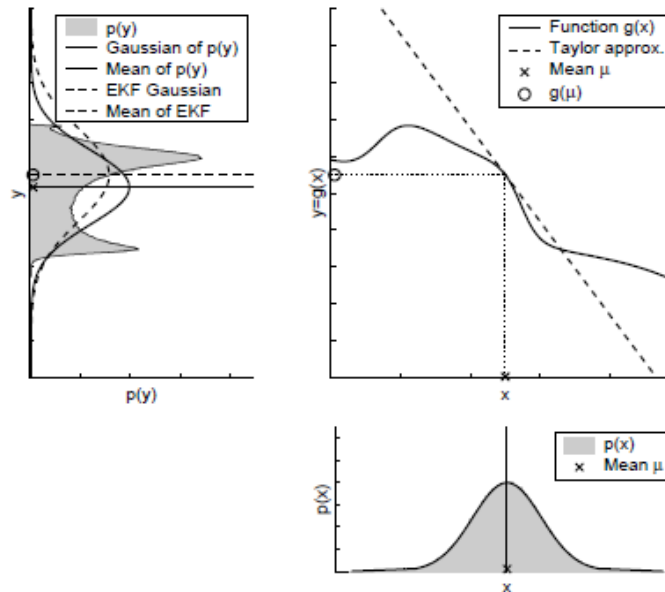


Figure 12 - Comparison of linearized Gaussian propagation vs. true propagation [27]

EKF-SLAM is very common. In order to implement such an approach, it is important to maintain a covariance matrix, which encompasses all landmarks. This

allows the EKF to develop pose and landmark estimates incrementally. As the number of landmarks grows, the matrix quickly becomes difficult to expand and update efficiently, and EKF-SLAM is very sensitive to outliers in landmark detection. After a single outlier measurement is incorporated into the covariance matrix, it cannot be corrected later if more information becomes available [34].

Newmann [48] proved that the EKF converges for linear SLAM problems in which the motion model and observation model are linear functions with Gaussian noise. However, this is not a novel concept, as a linear system makes the filter optimal and does not diverge as the nonlinear system can [1]. Nonlinearity can be a significant problem in EKF-SLAM. It leads to inevitable, and sometimes dramatic, inconsistency in solutions to EKF-based SLAM [1]. EKF algorithms essentially propagate first-order uncertainty in the coupled estimates of robot and map feature positions. There are various techniques for reducing computational complexity in large maps. These techniques have shown great success in enabling robots to estimate their locations accurately and robustly over large movement areas [38]. Overall, EKF-SLAM tends to converge. Experimental data shows landmark uncertainty monotonically decreasing over time [1]. This is exemplified in Figure 13, which shows a sample plot of landmark uncertainty over time. Initial observations have an initial variance that is dependent on the robot's location and pose. These uncertainties decrease over time, resulting in better mapping and localization. Each line represents the certainty of an observation and the state. As a landmark is first observed, its certainty starts large and decreases over time.

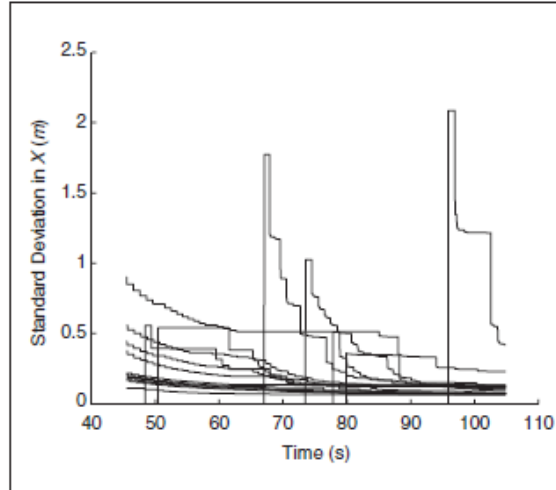


Figure 13 - The convergence in landmark uncertainty [1]

2.2.1.3. Unscented Kalman Filter

The Unscented Kalman Filter (UKF) uses a unique representation of a Gaussian random variable in N dimensions using $2N + 1$ samples, called sigma points. The representation utilizes the properties of the matrix square root and the covariance definitions to select these points in such a way that they have the same covariance as the Gaussian they approximate [45]. The UKF yields results comparable to a third-order Taylor series expansion of the state-model, while Extended Kalman Filters are only accurate to a first-order linearization. The Unscented Transform approach also has another advantage: noise can be treated in a nonlinear fashion to account for non-Gaussian or non-additive noises. The strategy for doing so involves propagation of noise through the functions by first augmenting the state vector to include noise sources [45]. Figure 14 illustrates how the UKF propagates the Gaussian distribution through a nonlinear function. Compared to Figure 11, this is a more accurate prediction scheme. In experiments, the UKF has yielded more accurate results than the EKF [45].

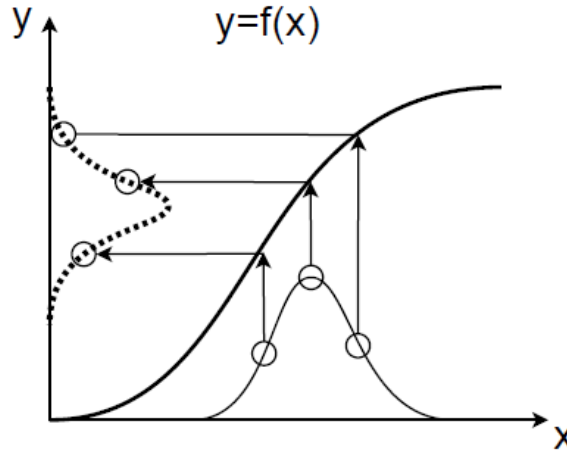


Figure 14 - Example of Unscented Transform with sigma points for Gaussian propagation [45]

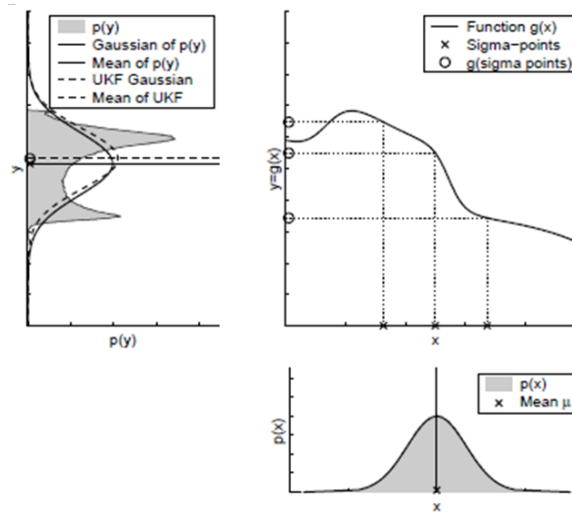


Figure 15 - Comparison of Unscented Transform with sigma points for Gaussian propagation vs. true propagation [27]

Many of the same issues plague the UKF as in the EKF. In addition, the computation cost is slightly greater than the EKF and still requires a squared order of calculations as compared to landmarks. Table 4 shows the complete UKF equations, this is compiled from several sources since there are some conflicting versions. The required space is again a squared order as compared to landmarks. The UKF suffers less from linearization, though it is not exempt. Finally, the UKF does not fully recover from poor landmarks, just as with the EKF.

System Equations:	$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \quad \mathbf{w}_k \sim N(0, \mathbf{Q}_k)$ $\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \quad \mathbf{v}_k \sim N(0, \mathbf{R}_k)$ $L = \dim(\mathbf{x}) \quad \alpha = 0.5 \quad \beta = 2 \quad \kappa = 0 \quad \lambda = \alpha^2 (L + \kappa) - L$
Calculate Sigma Points:	$\Sigma(\mathbf{x}, \mathbf{P}) := \begin{cases} \Sigma_0 = \mathbf{x} \\ \text{for } i=1 \dots L \\ \Sigma_i = \mathbf{x} + \left[\sqrt{(L + \lambda) \mathbf{P}} \right]_{(\text{Column } i)} \\ \Sigma_{i+L} = \mathbf{x} - \left[\sqrt{(L + \lambda) \mathbf{P}} \right]_{(\text{Column } i)} \end{cases}$
Prediction Phase:	$\Sigma_i^{k-1 k-1} = \Sigma(\mathbf{x}_{k-1 k-1}, \mathbf{P}_{k-1 k-1})$
Predict sigma points:	$\Sigma_i^{k k-1} = f(\Sigma_i^{k-1 k-1}, \mathbf{u}_k)$ $\mathbf{x}_{k k-1} = \frac{\lambda}{L + \lambda} \Sigma_0^{k k-1} + \sum_{i=1}^{i=2L} \frac{1}{2(L + \lambda)} \Sigma_i^{k k-1}$ $\mathbf{P}_{k k-1} = \mathbf{Q}_k +$
Predicted state and covariance:	$\left((1 - \alpha^2 + \beta) + \frac{\lambda}{L + \lambda} \right) (\Sigma_0^{k k-1} - \mathbf{x}_{k k-1}) (\Sigma_0^{k k-1} - \mathbf{x}_{k k-1})^T +$ $\sum_{i=1}^{i=2L} \frac{1}{2(L + \lambda)} (\Sigma_i^{k k-1} - \mathbf{x}_{k k-1}) (\Sigma_i^{k k-1} - \mathbf{x}_{k k-1})^T$
Update Phase:	$\Sigma_i^{k k-1} = \Sigma(\mathbf{x}_{k k-1}, \mathbf{P}_{k k-1})$
Observation sigma points:	$\tilde{\Sigma}_i^{k k-1} = h(\Sigma_i^{k k-1})$ $\hat{\mathbf{z}}_{k k-1} = \frac{\lambda}{L + \lambda} \tilde{\Sigma}_0^{k k-1} + \sum_{i=1}^{i=2L} \frac{1}{2(L + \lambda)} \tilde{\Sigma}_i^{k k-1}$ $\mathbf{S}_k = \mathbf{R}_k +$
Observation state and covariance:	$\left((1 - \alpha^2 + \beta) + \frac{\lambda}{L + \lambda} \right) (\tilde{\Sigma}_0^{k k-1} - \hat{\mathbf{z}}_{k k-1}) (\tilde{\Sigma}_0^{k k-1} - \hat{\mathbf{z}}_{k k-1})^T +$ $\sum_{i=1}^{i=2L} \frac{1}{2(L + \lambda)} (\tilde{\Sigma}_i^{k k-1} - \hat{\mathbf{z}}_{k k-1}) (\tilde{\Sigma}_i^{k k-1} - \hat{\mathbf{z}}_{k k-1})^T$ $\mathbf{P}_{xz} = \left((1 - \alpha^2 + \beta) + \frac{\lambda}{L + \lambda} \right) (\Sigma_0^{k k-1} - \hat{\mathbf{x}}_{k k-1}) (\tilde{\Sigma}_0^{k k-1} - \hat{\mathbf{z}}_{k k-1})^T +$ $\sum_{i=1}^{i=2L} \frac{1}{2(L + \lambda)} (\Sigma_i^{k k-1} - \hat{\mathbf{x}}_{k k-1}) (\tilde{\Sigma}_i^{k k-1} - \hat{\mathbf{z}}_{k k-1})^T$
Kalman gain:	$\mathbf{K}_k = \mathbf{P}_{xz} \mathbf{S}_k^{-1}$
Updated state:	$\mathbf{x}_{k k} = \mathbf{x}_{k k-1} + \mathbf{K}_k (\mathbf{z}_k - \hat{\mathbf{z}}_{k k-1})$
Updated covariance:	$\mathbf{P}_{k k} = \mathbf{P}_{k k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T$

Table 4 - Unscented Kalman Filter equations [27][49]

2.2.2. Information Filter

The Information Filter is similar to the Kalman Filter, but uses a different representation of a Gaussian. Instead of using the standard form with a mean and covariance, the filter uses the canonical parameters. The exact mechanics of the filter are not as important to this dissertation as the idea behind it. It is almost the opposite of a Kalman Filter, where statistics are accumulated and then when a position or map is necessary the solution can be extracted. This concept is known as a "lazy" filter, where actual results are computed in a batch process at the end. Table 5 shows the equations behind the Basic Information Filter. The conversion back to standard form is only necessary when the data is needed by another process.

System Equations:	$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$ $\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$ $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$ $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$
Information Filter	
Input: $(\xi_{k-1}, \Omega_{k-1}, \mathbf{u}_k, \mathbf{z}_k)$	
Output: (ξ_k, Ω_k)	
$\hat{\Omega}_k = \left(\mathbf{F}_k (\Omega_{k-1})^{-1} \mathbf{F}_k^T + \mathbf{R}_k \right)^{-1}$ $\hat{\xi}_k = \hat{\Omega}_k \left(\mathbf{F}_k (\Omega_{k-1})^{-1} \xi_{k-1} + \mathbf{B}_k \mathbf{u}_k \right)$ $\Omega_k = \mathbf{H}_k (\Omega_{k-1})^{-1} \mathbf{H}_k^T + \hat{\Omega}_k$ $\xi_k = \mathbf{H}_k (\Omega_{k-1})^{-1} \mathbf{z}_k + \hat{\xi}_k$	
Conversion:	$\mathbf{x} = \Omega^{-1} \xi$ $\mathbf{P} = \Omega^{-1}$

Table 5 - Basic Information Filter equations [27]

Variations of the Information Filter exist, just like variations of the Kalman Filter. The Extended Information Filter uses a linearization of the system just like the EKF;

however, a slight change in the equations is required, but this is straight straightforward. The Information Filter is not an active filter, thus it gives way to other versions of SLAM. The Sparse Extended Information Filter (SEIF) is interesting as it compresses the matrix representations to increase computational speed. The next section covers sparsification, which is the key component of SEIF. GraphSLAM, a popular algorithm for large maps is a modified Extended Information Filter.

2.2.2.1. *Sparse Extended Information Filter*

The Sparse Extended Information Filter (SEIF) makes use of the fact that the off-diagonals of the information matrix Ω are usually near zero. The process of setting these values to zero is called sparsification. With the information matrix now sparse, very efficient update procedures for information estimates can be performed with relatively little loss in optimality or accuracy of the maps produced [50]. Sparsification allows smaller maps to be generated and subsequently fitted into the global map. This technique mitigates some of the remaining drift problems in SLAM wherein features are tracked poorly in the resulting distance. Figure 16 shows how modifying the information matrix changes the constraints between the robot and past landmarks. Figure 17 shows sample sparse maps taken at (a) and at (b), which combine to form the larger map (c). Producing (c) is done in an optimal, error-minimizing fashion.

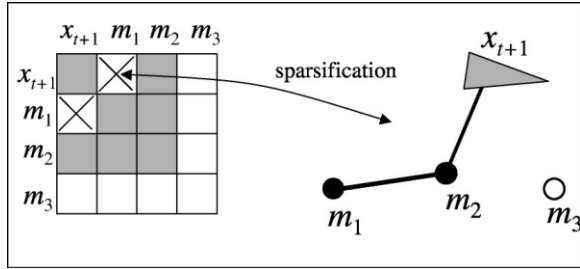


Figure 16 - Graphical representation of sparsification from Information Matrix to map [27]

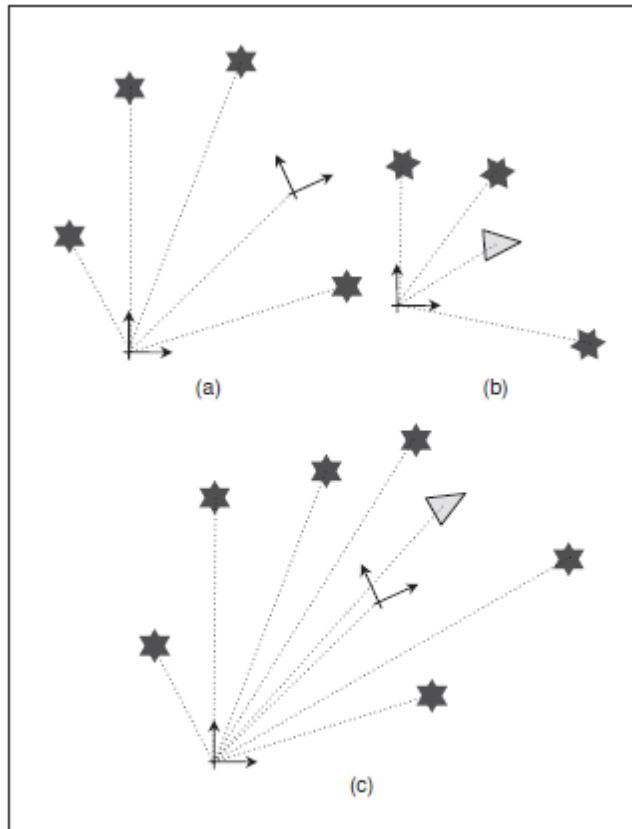


Figure 17 - Sparsification with sub-maps. Maps (a) and (b) are combined to form map (c) [50]

2.2.2.2. GraphSLAM

GraphSLAM is an offline or "lazy" SLAM method that collects data into an information matrix, much like the Information Filter. Similar to SEIF, GraphSLAM reduces the information matrix, however this is done through removing landmarks, not sparsification [27] [51]. Figure 18 shows sample constraints connecting robot poses and

landmarks. The information matrix is conceptually easier to show these constraints, the figure shows the matrix elements that correspond to constraints between the last two robot poses and from the last landmark to the last pose.

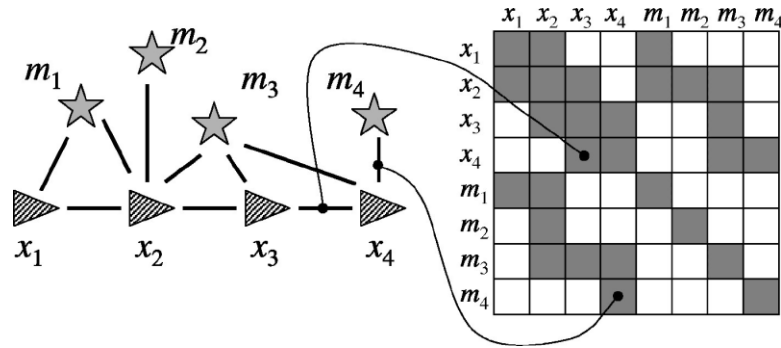


Figure 18 - Conceptual diagram of GraphSLAM constraints in the Information Matrix [51]

GraphSLAM is updated in a particular way to form such an ideal information matrix. This pattern allows constraints and entire landmarks to be removed. Figure 19 shows the removal of the first landmark. This is done by applying the inverse of the landmark's certainty (located on the diagonal of the matrix) and multiplying by the connecting constraints. After these values are removed, the row and column relating to the landmark are completely removed [27] [51].

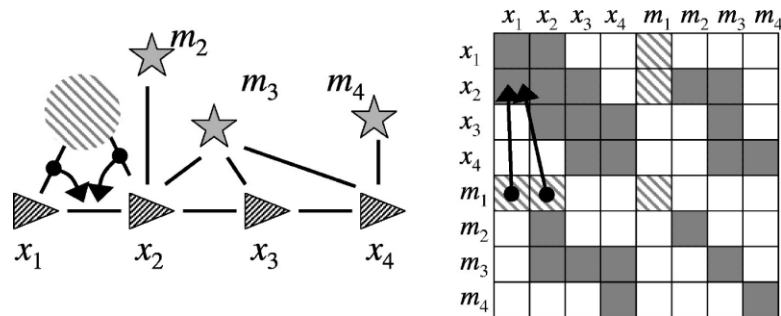


Figure 19 - Conceptual diagram of how GraphSLAM removes landmarks [51]

GraphSLAM reduces the information matrix, landmark by landmark. After a reduction of several landmarks, an intermediate solution is calculated using a gradient

descent-like technique. The system is recalculated, reduced, and solved repeatedly until it converges. This is all done after all data has been collected [27] [51].

2.2.3. Particle Filters

Particle filters provide another model estimation technique based on simulation. They are typically used to estimate Bayesian models and are much like an on-line version of a Markov chain Monte Carlo batch method [52]. Particle filters are often used as an alternative to the EKF or UKF. With sufficient samples, particle filters have the advantage of approaching the Bayesian optimal estimate and therefore can be more accurate than either the EKF or UKF [53]. The approaches can also be combined by using a version of the Kalman Filter as a proposal distribution for the particle filter. This approach uses a particle filter in which each particle carries an individual map of the environment [52].

The main drawback is that the number of particles required in particle filtering grows exponentially with the size of the state vector [53]. The particle filter outperforms the EKF with regard to outliers. However, as previously mentioned, it scales poorly with respect to the dimensionality of the state [34]. To overcome the computational cost, other particle filters have been developed, such as the Rao-Blackwellized Particle Filter (RBPF), which reduces the number of required particles. Particle filters also have issues of depletion, where random selections of particles are no longer in the area of the correct solution. Furthermore, particle filters do not correct the state, only the expected distribution is modified. This is a misconception of particle filters. A particle filter

simply selects a random robot pose and tests its validity, but never does the filter actually modify the pose, only the map.

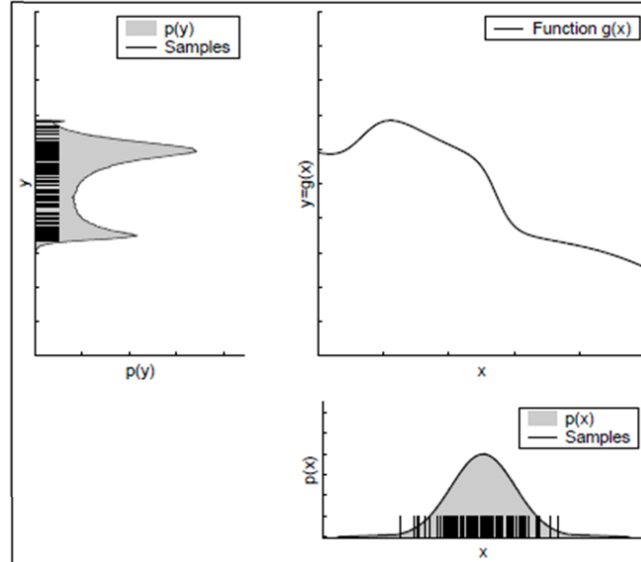


Table 6 - Particle Filter propagation of a Gaussian [27]

2.2.3.1.

Rao-Blackwellized Particle Filter

Rao-Blackwellized Particle Filters (RBPF) have been used in a number of SLAM applications [52][54][55][56]. The RBPF was introduced as an effective means to solve the SLAM problem. In a standard particle filter, each particle carries an individual map of the environment. The key question is how to reduce the number of particles, which is what the RBPF does [52]. The high dimensional state-space of the SLAM problem makes direct application of particle filters computationally infeasible. However, it is possible to reduce the sample space by applying Rao-Blackwellization [1]. The RBPF reduces computational unwieldiness by factoring the state variables such that, by sampling over a subset of them, it is possible to marginalize out the remaining ones [34]. The filter combines the approximate technique of particle filtering for some variables

with exact filtering (based on the values of the particle-filtered variables) for the remaining variables [55].

In more precise terms, by applying Rao-Blackwellization, a joint state is partitioned according to the product rule $P(x_1, x_2) = P(x_2 | x_1)P(x_1)$ and if $P(x_2 | x_1)$ can be represented analytically, only $P(x_1)$ needs be sampled. The joint distribution is represented by the set $\{x_1^{(i)}, P(x_2 | x_1^{(i)})\}_i^N$ and statistics such that the marginal $P(x_2) \approx \frac{1}{N} \sum_i^N P(x_2 | x_1^{(i)})$ can be obtained with greater accuracy than is possible by sampling over the joint space [1]. This sampling and approximation can be more accurate than Kalman Filter approaches [53].

General SLAM approaches that use the RBPF assign each particle a single hypothesis about the robot's pose. The hypothesis about each pose is based on the pose history. Each particle uses an exact filter to obtain a map [55]. What this means in practice is that the trajectory is represented by weighted samples and the map is computed analytically. The map accompanying each particle is composed of independent Gaussian distributions. Recursive estimation is performed by particle filtering for the pose states and using an EKF for the map states.

One research group [55] presents an adaptive re-sampling technique that maintains a variety of particles but reduces the amount of redundant particles. This enables the algorithm to learn a map accurately while reducing the risk of particle depletion. The authors comment that the common approach with these filters is to use a smoothed likelihood function. However, this prevents particles close to the meaningful

area from getting a too-low importance weight and discards useful information gathered by the sensor, often leading to less accurate maps in the SLAM context [55].

2.2.4. FastSLAM 1.0 and 2.0

FastSLAM uses a RBPF to sample a path. Each particle contains its own map, which consists of N Extended Kalman Filters [47]. The FastSLAM algorithm, introduced by Montemerlo et al., marked a conceptual shift in the design of recursive probabilistic SLAM. Previous efforts focused on improving the performance of EKF-SLAM while retaining its essential linear Gaussian assumptions. FastSLAM, with its basis in particle filtering, was the first to represent the nonlinear process model and non-Gaussian pose distribution directly [1]. Additionally, FastSLAM uses RBPF with tree structure, which results in a performance on the order of $O(M \log N)$ versus a normal structure with $O(MN)$. M is the number of new particles and N is the number of current landmarks [34]. FastSLAM still linearizes the observation model but does it such that it reasonably approximates range-bearing measurements when the vehicle's pose is known [1].

Figure 20 shows a sample trajectory of a robot using FastSLAM. The landmarks are indicated with white stars, and the actual path is depicted by the black dotted line. The white triangle represents the actual position. The blue stars represent measured landmarks, or particles, and the ellipsoids show uncertainty estimates for position. The red lines indicate landmark measurements and the yellow triangle represents the perceived location.

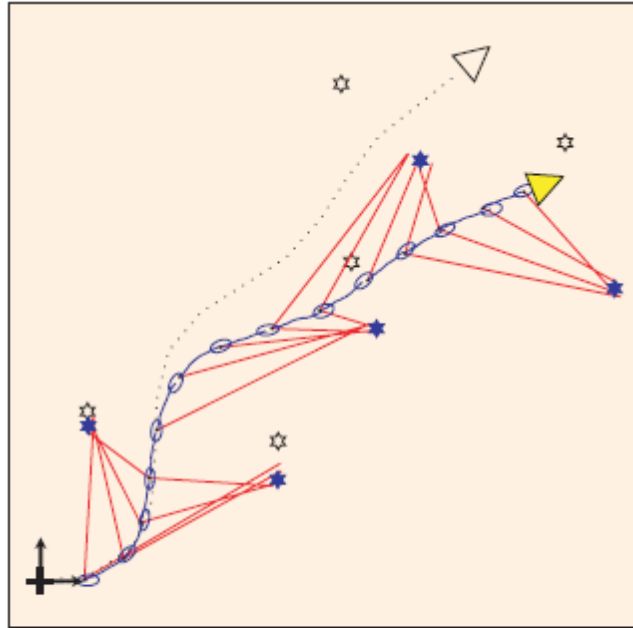


Figure 20 - Sample robot trajectory using the FastSLAM approach [1]

The inventors of FastSLAM revised the initial algorithm and created FastSLAM 2.0. This approach still uses a particle filter to predict the robot's pose. However, instead of having the probability distribution rely only on the motion estimate, the newer approach also incorporates the most recent sensor measurement. Such an approach is less wasteful with its samples than the original FastSLAM algorithm, especially in situations where the noise in motion is high relative to the measurement noise [47]. In both FastSLAM 1.0 and 2.0, it is unknown how many particles are needed for convergence. The number of particles needed is suspected to be, at worst, exponential compared to the size of the map [47].

2.2.5. Monocular SLAM

Monocular SLAM is specific to the use of a single camera (for ranging) accompanied by various other sensors. Using a single camera for SLAM poses many

interesting problems, such as developing a method for obtaining ranging, information using the camera motion (structure from motion), and developing robust techniques for finding good landmarks. Monocular SLAM can provide insight when applied to stereo camera systems, since any monocular system can also be used on a stereo camera system without any modifications.

MonoSLAM was the first successful application of SLAM with mobile robotics in the “pure vision” domain of a single un-actuated camera. The core of the MonoSLAM approach is the online creation of a sparse, but persistent, map of natural landmarks within a probabilistic framework. The creators of MonoSLAM showed that a small set of landmarks could provide a very accurate SLAM reference if carefully chosen and spread [46]. The key to MonoSLAM is use of the probabilistic feature-based map, which represents, at any instant, a snapshot of the current estimates of the camera’s state and all features of interest, along with their respective uncertainty. Interestingly, MonoSLAM’s accuracy is significantly improved by exchanging per-pixel angular resolution for increased field of view. Normally, one would think that better pixel resolution would achieve better results, yet camera and map estimates are much better constrained when features are viewed at very different angles at the same time [46]. Figure 21 shows more samples from MonoSLAM with camera snapshots and 3D landmark patch projections.

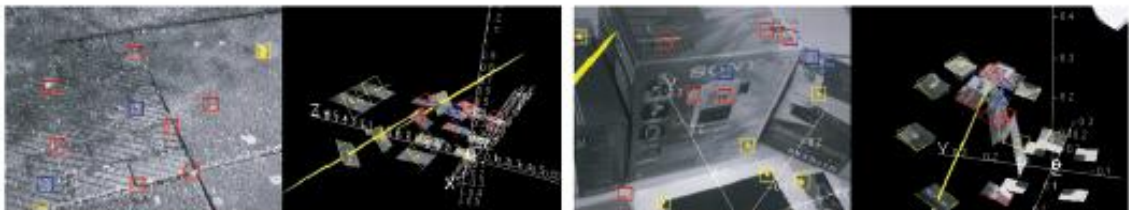


Figure 21 - Sample snapshots and projections from MonoSLAM [46]

Jensfelt, et al, created a monocular SLAM approach using the EKF and SFM [4]. They used a Harris corner detector with Laplacian pyramids for feature detection and tracking. With a single camera, it is not enough to match a single landmark descriptor against the database to determine pose. However, by including all the descriptors from the scene where the descriptor in question was found, the matching becomes very robust. The idea is to let the SLAM estimation lag behind N frames, use these N frames to determine which points make good landmarks and to find an estimate of their 3D location. Another monocular SLAM approach was created by [36]. The authors included some examples of their SLAM implementation, which looks very much like MonoSLAM.

Monocular based SLAM has a few inherent flaws. For one, classic EKF-SLAM algorithms cannot be directly applied. The addition of a feature to the stochastic map requires a full Gaussian estimation of its state. This means a special landmark initialization process must be used, which combines at least two observations of the same feature from sufficiently separated robot poses [36][42]. Many authors use the Mahalanobis distance and the Chi-Squared test to match observed features with landmarks in the map. This is not appropriate in the bearing-only case, because a single observation can be matched with all the landmarks that are near the corresponding line in the map. The 2D feature tracking used in MonoSLAM only allows relatively slow camera motions, and feature re-acquisition after periods of neglect is not allowed. Features typically survive for 20 to 40 frames; afterward, they are replaced in the state vector by others. This means that, as a localization method, motion drift would eventually enter the system [38].

2.2.6. Stereo Vision SLAM

Stereo vision SLAM uses two or more cameras in conjunction with normal SLAM methods. Stereo vision SLAM has inherent advantages compared to monocular approaches. Even if monocular approaches produce perfect estimates, the stereo approach gives more information, because the overall scale of the motion is immediately and instantaneously known based on the separation of the stereo cameras. Furthermore, the case of slow or no motion can be dealt with easily without special handling [57]. Stereo vision allows for easy calculation of visual odometry and ego-motion and aids in navigation.

Garcia and Solanas used stereo vision to produce full 6 DoF ego-motion estimates (3D translation, roll, pitch, and yaw) [44]. Their system was similar to Structure-From-Motion (SFM) algorithms. The problem with SFM is that both the 3D geometry and camera movements are only recovered up to a scale factor. Thus, real distances are lost. This drawback was corrected by manually calibrating the robot with respect to a target of known size at the beginning of the process. The approach used the Harris corner detector on images in order to determine the corner strength of every pixel [23]. Removing false features is very important in stereo vision. The authors used various approaches to reduce the amount of false matching in stereo imaging. Pixels whose strength was above an experimental threshold were chosen as visual landmarks. Points were further filtered on calculated distances from the mean or median. A point was filtered out if the colors did not match. The system generated a volumetric 3D model of the explored workspace. Results were still very noisy, but an octree was used to remove some redundant points. The noisy results might have occurred because the authors used a pinhole camera model

[44]; pinhole models ignore the optics of the camera. Performing camera calibration that accounts for lens distortion may improve the results.

Stereo Vision SLAM, like all SLAM approaches, is reliant on the quality of its sensor input. The quality, in this case, is determined by the accuracy of the stereo correspondence points. Since stereo cameras have a fixed separation, stereo vision sensors contain more noise, produce inaccurate estimates, and have a trade-off between angular field of view and depth perception [58]. Typical stereo systems capture images at a resolution of 320 x 240 pixels. The images are then used to produce a disparity map [58]. SFM approaches can be easily applied to stereo applications for aiding in environmental analysis improvements. The following sections of this area will discuss some specific types of stereo vision SLAM such as EKF approaches, particle filter approaches, and approaches that use SIFT.

2.2.6.1. Stereo SIFT SLAM

SIFT will be further explained in a section 2.4.2 on page 57. The basic concept behind SIFT is that it produces key features on an image and encodes those features in a vector in a way that facilitates comparisons. SIFT features are relatively invariant to location and lighting yet distinct enough for use in object recognition [59].

In [60], the authors implemented 3D Stereo Vision SIFT SLAM. Stereo correspondence was implemented by matching SIFT features in the stereo pair of images. Instead of building a map continuously, the authors built multiple 3D sub-maps, which were subsequently merged together. The system ran only at 2 Hz on a Pentium III 700 MHz processor. Sub-maps were built every 30 frames or, in essence, every 15 seconds.

When the robot was facing a scene with very few SIFT landmarks, it might not have been able to localize itself globally using just the current frame. To achieve more robustness, a small sub-map of a local region from multiple frames was used instead. It was then aligned to the database map. Experiments showed that global localization could be achieved with just the current frame data in feature-rich environments. The distinctive SIFT features enabled global localization.

A SLAM system using stereo vision and a Rao-Blackwellized particle filter was presented in [54]. Visual features were detected using Difference-of-Gaussians, and stereo features were matched using the SIFT descriptor. In this work, two important problems arose that were blockers for a real-time implementation, but were not solved. The first unresolved problem was that there were a large number of detected features, which made the approach inappropriate for large-scale and textured environments. The second issue involved the management and correspondence of SIFT features, because matching was performed by one-to-all comparison. It was mentioned that a better strategy would involve implementing KD-trees and using a limited number of features.

In [56], the authors introduced a SLAM approach known as σ SLAM. The approach used stereo vision and the Rao-Blackwellized particle filter (RBPF). Also, SIFT features were used as landmarks. The authors constructed maps of 3D point-landmarks identified by their visual appearance. The particle filter approach utilized a mixture proposal distribution. It accurately tracked the position of the robot and the landmarks over long trajectories. In addition, the approach did not depend on robot odometry in the particle filter's proposal mechanism. Instead, the proposal distribution relied on visual odometry, which made the transition towards a SLAM solution for robots

performing 3D motion much easier. The approach constructed 2D occupancy-grid maps from stereo vision, which can be used for path planning and obstacle avoidance [56]. The approach implemented a Bayesian framework based on the RBPF. The entire sequence ran on a Pentium Xeon 3.2 GHz computer. The average time over all frames was 1.5 seconds. A large portion of the time, 0.35 seconds, was spend on SIFT extraction and matching. As suggested in [54], this approach organized all SIFT keys with a KD-tree data structure, allowing for faster access time. The approach was fast, but it was not fast enough to be real-time.

2.3. Cooperative Control and Coverage

Cooperative control and map coverage are out of the scope of this dissertation. Though the topic is cooperative SLAM, the control algorithm is irrelevant to the mapping. However, for completeness, this section is included to present possible control and coverage techniques.

2.3.1. Cellular Decomposition

There are many forms of path planning for map coverage and discovery. Classic approaches include methods such as “follow the wall” and random motion. Other approaches use mapping techniques like cellular decomposition [61]. Cellular decomposition is very common for SLAM applications in which the world is represented as a grid of boxes and there is a probability that obstacles or walls will reside in those boxes. There are Spanning Tree Covering algorithms that follow a spanning tree to

subdivide an area into disjoint cells. There are coverage approaches that use zigzagging techniques with recursion for inlets. Also, an algorithm by [61] used Boustrophedon decomposition, which makes trapezoidal cells that account for clumping. Figure 22 shows an example of Boustrophedon decomposition, which would normally be mapped by three adjacent cells.

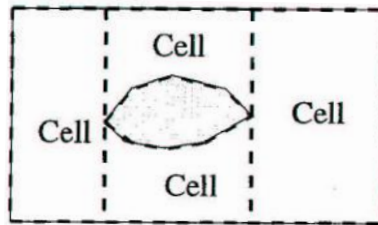


Figure 22 - Boustrophedon decomposition [61]

2.3.1.1. Spanning Trees

Another exploration algorithm is the least recently visited (LRV) algorithm. One research group [62] demonstrated the following two properties: (1) LRV is complete on graphs, and (2) LRV is optimal on trees. The authors presented experimental conjectures for LRV on a regular square cube lattice graph, and empirically compared its performance to other graph exploration algorithms. It was shown on a square lattice that, with an appropriately chosen order, LRV performs optimally. LRV works by analyzing a map to locate the “frontiers” between the free and unknown space. Exploration then proceeds in the direction of the closest “frontier.” Figure 23 shows a sample map as explored by LRV and the associated tree structure.

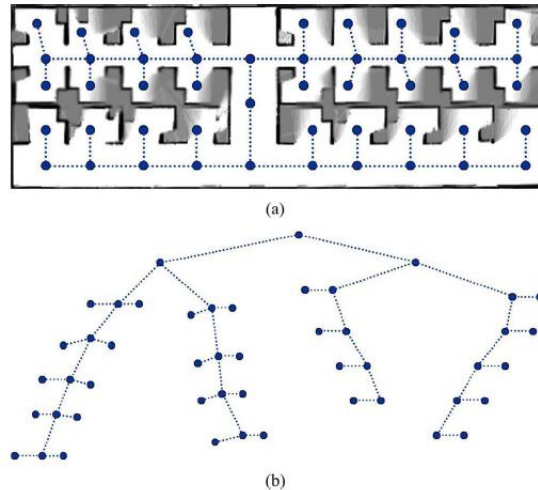


Figure 23 - Map explored using LRV shown in (a), tree representation of map shown in (b) [62]

2.3.2. Server-based Swarm

Swarm robotics usually takes the form of many low-level drone robots that perform tasks either in a distributed manner or with a server as a coordinator. In [63], the authors described swarm robotics in abstract terms. Their communication architecture was centralized, with a server controlling all drones. The authors showed that, in the particular case of an abstraction based on centroid and variance, swarm cohesion, inter-robot collision avoidance, and environment containment could be specified and automatically guaranteed. The framework was highly theoretical, with robots abstracted as fully actuated on a 2D plane. The system was neither implemented nor tested. The framework in this system is an interesting one, due to its structure.

2.3.3. Pseudo-Potential Fields

Another way of controlling a large group of robots involves the notion of potential fields. In essence, an energy equation is created such that the robots attempt to

minimize the energy equation and behave in a desirable manner. This area is dubbed “Pseudo-Potential Fields,” because the energy equation and subsequent potential fields are artificially created to elicit certain behaviors from a group of robots.

In [64], a cooperative control system that could perform perimeter patrolling was described in detail. In a perimeter detection task, a robotic swarm located and surrounded a substance while dynamically reconfiguring as additional robots located the perimeter. A decentralized, cooperative hybrid system was created. The system was inspired by biological systems that use potential fields. Potential fields were used for cooperative control. The authors developed a method that used artificial potentials and virtual bodies to make the robot network form regular polygons upon uniformly surrounding a target. Figure 24 shows three robots in an experimental setup. The robots were able to find the perimeter of the blue region and surround the target in a uniform manner. Figure 25 shows the position and trajectories of the three robots as they surrounded the target, as shown in Figure 24.



Figure 24 - Indoor experimental setup for perimeter detection [64]

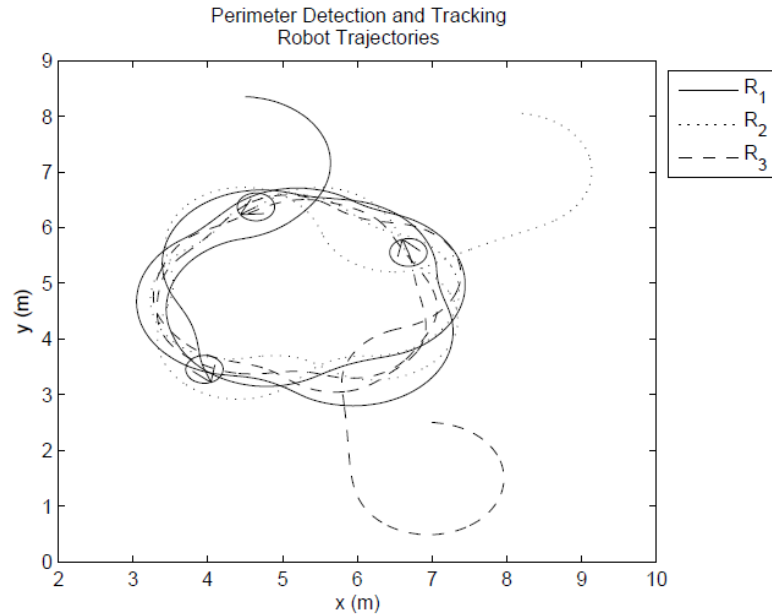


Figure 25 - Three robots defining a perimeter [64]

In [65], the authors addressed the challenge of controlling a network of agents such that the resulting motion always preserved the connectivity property of the network. In particular, the connectivity condition was translated to differentiable constraints on individual agent motion. This was done by considering the dynamics of the Laplacian matrix and its spectral properties. Artificial potential fields were used to drive the agents into configurations away from the undesired space of disconnected networks. In the meantime, the agents avoided collisions with each other. The authors noted that the second smallest eigenvalue of the Laplacian served as a measure of stability and robustness in the dynamic networked system. It is important that the potential field is differentiable such that gradient descent techniques can be applied. Table 7 shows the set of functions used to define the artificial potential field for maintaining connectivity and collision avoidance. The function $x_i(t)$ represents the position of robot i at time t . The variable d represents the connectivity distance. Figure 26 shows a sample plot of the

equation in Table 7 with $d = 0.5$. The field does not allow robots to approach each other or break network connectivity.

$$\beta_{ij}(x(t)) = \left(1 - \mu \frac{(\|x_i(t) - x_j(t)\|^2 - d^2)^2}{1 + (\|x_i(t) - x_j(t)\|^2 - d^2)^2} \right)^\rho$$

$$\mu = (1 + d^4)/d^4$$

$$\rho = (1 - \text{sign}(\|x_i(t) - x_j(t)\| - d))/2$$

Table 7 - Pseudo-potential equations for maintaining connectivity and collision avoidance [65]

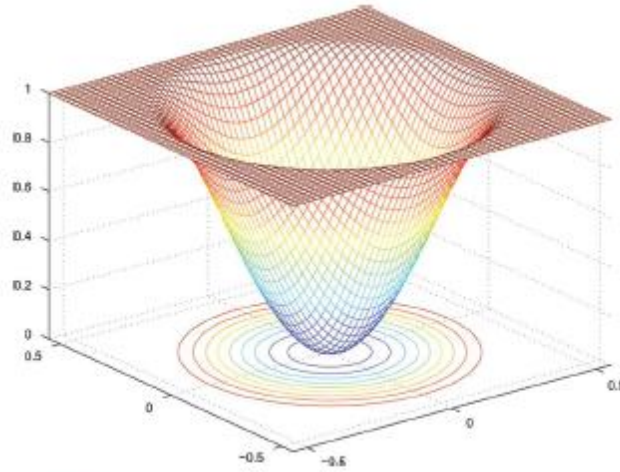


Figure 26 - Plot of the equation in Table 7 with an absolute difference function [65]

2.3.4. Graph Coordination

Graph coordination uses the shape and lengths of edges in graphs to make intelligent control decisions. In cooperative control, each robot is represented by a node in the graph. Typically, the connecting lines represent distances between robots. However, multiple other set-configurations exist.

As mentioned in the previous paper, cooperative control requires robots to remain in radio contact. In [66], the authors addressed the issue of cooperative control that

preserves connectedness. The agreement problem (or consensus problem) is concerned with finding decentralized strategies that achieve convergence to a common value. The agreement is typically achieved through a nearest-neighbor-like protocol $\dot{x}_i(t) = -k_i(t) \sum_{j \in \text{nbhd}_i(t)} \alpha_{ij}(t) (x_i(t) - x_j(t))$ where x_i is the state vector of the i^{th} robot. The authors used Lyapunov functions to analyze the stability of the network with a decentralized control law. They showed that the above law is, in fact, asymptotically stable. A collection of graph-based nonlinear feedback control laws were studied for distributed multi-agent systems. The nonlinear feedback laws were based on weighted graph Laplacians. They were proven able to solve the rendezvous and formation-control problems while ensuring connectedness.

2.3.5. Optimal Cooperative Coverage

Given a group of cooperative robots, there exist ways to optimally explore, cover, and rendezvous. Researchers, such as Cortés and Bullo, showed that given a set of networked robots with only line-of-sight sensors, that exploration, coverage, and rendezvous can be accomplished using the known geometry of the environment, or the direct sight of neighbors[67][68]. The exact details of these methods are far from the scope of this dissertation; however, it is sufficient to note that the primary mechanism for movement is based on gradient descent, and that the researchers showed the navigation functions to be globally Lipschitz and asymptotically converge[67][68]. Figure 27 and Figure 28 show examples of cooperative exploration and cooperative rendezvous using the gradient descent techniques described by Cortés and Bullo.

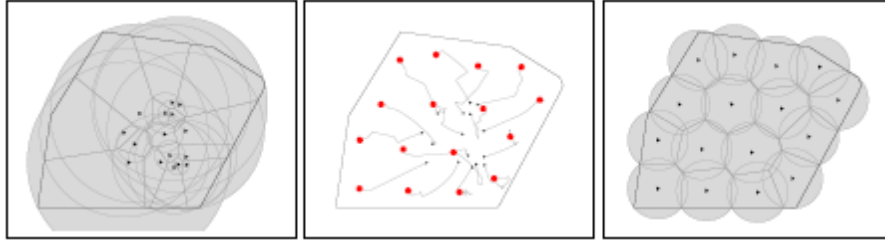


Figure 27 - Cooperative exploration example [67]

In Figure 27, the robots start clustered together, this is shown in the left image. Each robot spreads away to fully cover the map along paths ending where the red dots are showing after 16 iterations. The right image shows the final positions of the robots and each of them having the same coverage circle.

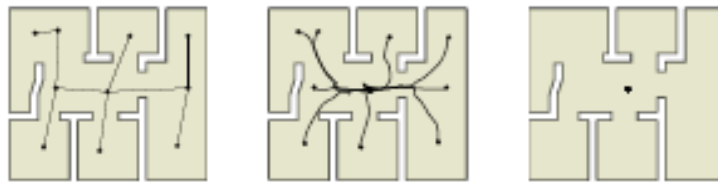


Figure 28 - Cooperative rendezvous example [68]

In Figure 28, the robots start in different sections of a non-convex space, using only line-of-sight sensors. The robots form the graph shown in the right image. The center image shows the path each robot follows using gradient descent. The right image shows the final position of all the robots in the graph.

2.4. Landmark and Feature Analysis

This section will be primarily devoted to image feature extraction. The image features examined include SIFT, SURF, and Harris Corners. The last section includes a discussion of the Lukas-Kanade Feature Tracker.

2.4.1. Scale Invariant Feature Transform (SIFT)

Scale Invariant Feature Transformation (SIFT) is one of the best-known advances in computer vision [59]. Features are invariant to image scaling, translation, rotation, partially invariant to illumination changes, and affine transformations or 3D projection. Features are found using a Difference-of-Gaussians (DoG). These features are similar to neurons located in the brain's inferior temporal cortex, which is used for object recognition in primate vision. Features are efficiently detected through a staged filtering approach that identifies stable points in scale space. Image keys are created that allow for local geometric deformations by representing blurred image gradients in multiple orientation planes and at multiple scales. The keys are used as input to a nearest neighbor indexing method that identifies candidate object matches. Experimental results show that robust object recognition can be achieved in less than two seconds, even in cluttered, partially occluded images [59].

SIFT gets mixed reviews when used for SLAM applications. One major complaint is that the algorithm takes too long to run [8][69]. When Lowe himself used SIFT as a means for conducting stereo vision SLAM, the system ran at 2 Hz on a Pentium III 700 MHz processor [60]. Commercial robotics companies, such as General Dynamic Robotic System, criticize SIFT for being too computationally expensive. They also insist that SIFT is not worth the processor time consumption for fielded robots that perform visual tracking [69]. Some researchers criticize SIFT for producing too few features for tracking [42]. However, the positive aspect of SIFT is its ability to produce distinctive features from natural landmarks [31][34][43][60]. The distinctiveness of SIFT

allows SLAM algorithms to perform global localization more easily and allows closing-the-loop approaches to work robustly [31] [60].

2.4.2. Speeded Up Robust Features (SURF)

Speeded-Up Robust Features (SURF) was loosely based on SIFT. Its creators, Bay, Ess, Tuytelaars, and Van Gool, designed SURF using integral images for faster computation. They based image points on Hessian matrices and used Haar-Wavelets for feature decomposition [25]. SURF approximated, and even outperformed, previously proposed schemes with respect to repeatability, distinctiveness, and robustness. SURF also computed and compared much faster than other schemes, allowing features to be quickly extracted and compared. This speedup was achieved by relying on integral images for image convolutions. Figure 29 shows a recall-precision comparison of SURF, SIFT, and GLOH. SURF outperformed the others by a fair margin. More importantly, it performed much faster.

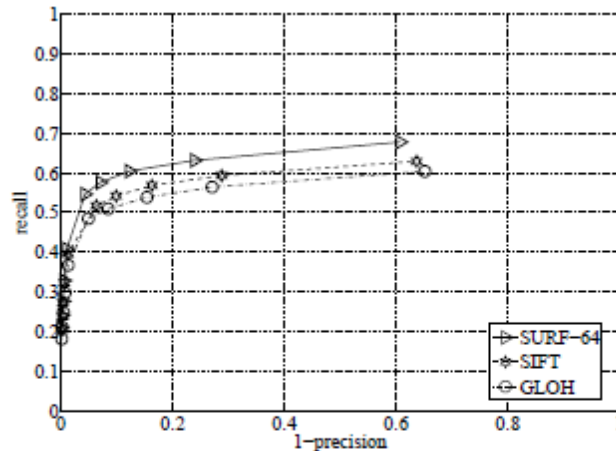


Figure 29 - Comparison of recall-precision for SURF, SIFT, and GLOH on eight images from Mikolajczyk's database [25]

2.4.3. Harris Corners

In image tracking, corners are usually the easiest to track, as they are unique in two dimensions of an image and provide locally unique gradient patterns. They help in feature recognition and tracking over small movements [23]. Several SLAM researchers used Harris Corners to track features [8][12][21][26][70]. Harris corner detection was primarily conducted on two-dimensional grayscale images. For the presented mathematical equations, the image intensity will be identified as I , and $I(x, y)$ would be the intensity at pixel location x, y . The corner detection method looks at an image patch around an area centered at (x, y) and shifts it around by (u, v) . The method uses the gradients around this patch. First, we calculate the weighted sum of square difference between the original patch and the translated patch, which is label (1) in Table 8. $I(u+x, v+y)$ is approximated by a Taylor expansion (label 2). Approximating the sum arrives at the equation labeled (3). Under label (3), I_x and I_y are partial derivatives of I in the x and y directions. These derivatives can be written in matrix form as shown in label (4). The matrix A can be expressed as a sum of weighted local gradients. The matrix A is known as the Harris matrix [23].

$$\begin{aligned}
S(x, y) &= \sum_u \sum_v w(u, v) (I(u, v) - I(u + x, v + y))^2 & (1) \\
I(u + x, v + y) &\approx I(u, v) + I_x(u, v)x + I_y(u, v)y & (2) \\
S(x, y) &\approx \sum_u \sum_v w(u, v) (I_x(u, v)x + I_y(u, v)y)^2 & (3) \\
S(x, y) &\approx (x \ y) A \begin{pmatrix} x \\ y \end{pmatrix} & (4) \\
A &= \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} & (5)
\end{aligned}$$

Table 8 - Derivation of the Harris Corner Detector [23]

A corner (or any interest point) is characterized by a large variation of S in all directions of the vector $\begin{bmatrix} x \\ y \end{bmatrix}$. By analyzing the eigenvalues of A , the image patch characterization can be expressed in a simple way: A should have two "large" eigenvalues for an interest point. Based on the magnitudes of the eigenvalues α and β , if $\alpha \approx 0$ and $\beta \approx 0$ then the pixel (x, y) has no features of interest. When an edge is found, $\alpha \approx 0$ and β has some large positive value. If a corner is found, α and β will both have large positive values. This characterization is shown in Figure 30, which was taken directly from Harris and Stephens' paper on corner detection [23].

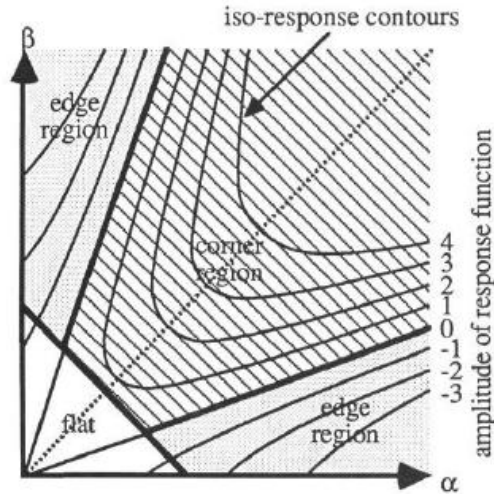


Figure 30 - Auto-correlation principal curvature space-heavy lines five corner/edge/flat classifications; fine lines are equi-response contours [23].

Harris and Stephens note that exact computation of the eigenvalues was computationally expensive, since it required the computation of a square root. In an alternative approach, the algorithm would not have to compute the eigenvalue decomposition of the matrix A . Instead, it would be sufficient for it to evaluate the determinant and the trace of A to find corners or other interest points [23].

In [42], the authors used Harris corners to perform stereo matching. They used an equation very similar to the one in label (3) in Table 8. Figure 31 shows a sample screenshot from the SLAM approach these authors took. The image shows how Harris corners were used for both motion tracking at the pixel level and for stereo matching. In [57], the authors also used Harris corner detection in all images and tracked features over time as well.

In the commercial robotics world, Harris corners are used by state-of-the-art positioning and location algorithms. For example, General Dynamic Robotic System

uses Harris corners in various image-processing algorithms for asset tracking, visual odometry, and image stabilization [69].

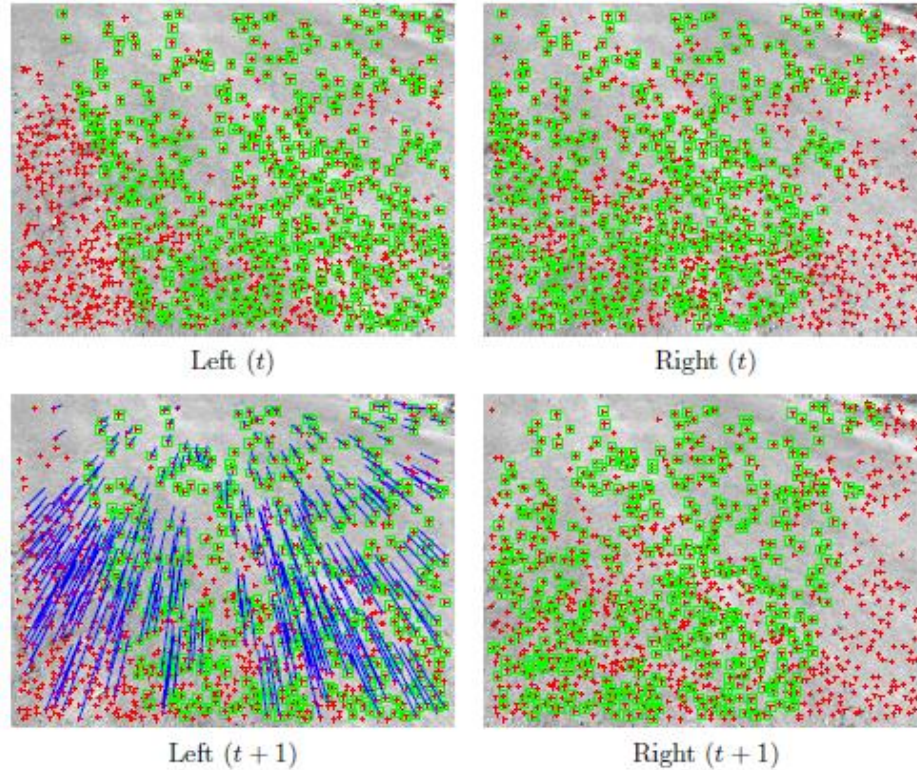


Figure 31 - Sample screenshot of stereo feature tracking over time using Harris [42]

2.4.4. Lucas-Kanade feature tracker

Tracking features across images is vital in visual odometry and in some forms of visual landmarks. SIFT features are tracked using the nearest neighbor comparison as outlined by Lowe [59]. Similarly, SURF features are also tracked using a nearest neighbor comparison. Harris corners, on the other hand, do not have such distinctive descriptors. They only detect features that are easy to track. The Lucas-Kanade Feature Tracker (LK Tracker) is a form of Lucas-Kanade optic flow. The qualities that make the LK Tracker so desirable are its speed, its robustness, and its sub-pixel accuracy. The two

most important components of any feature tracker are its accuracy and robustness [22][24]. The accuracy component of a tracker is related to its sub-pixel accuracy, which will be outlined later. The robustness component relates to tracking sensitivity with respect to changes in lighting, size of an image, and motion. In particular, in order to handle large motions, it is intuitively preferable to pick a large integration window.

The LK Tracker is used in industry today. General Dynamic Robotic Systems uses this method for its robustness and speed [69]. The company combines Harris corners with a modified LK Tracker to track up to 5,000 features in an image. The LK Tracker might be older than SIFT or other new tracking methods, but it is still useful in the industry as a performance tracker.

A common implementation of the LK Tracker involves inverted pyramids. Inverted pyramids transforms an image into a series of images, each image has half the width and height of the last layer, reducing resolution by averaging pixels, starting with the original image. The inverted pyramid means that an algorithm will start at the smallest image, settle to region of interest, and then proceed to the next layer given the bounded region. An iterative implementation of the Lucas-Kanade optical flow provides sub-pixel local tracking accuracy [22][24]. An outline of the LK tracker with inverted pyramids is shown in Table 9.

Goal: Let u be a point on image I . Find its corresponding location v on image J	
• Build pyramid representations:	$\{I^L\}_{L=0,\dots,L_m}$ and $\{J^L\}_{L=0,\dots,L_m}$
• Initialization of pyramidal guess:	$g^{L_m} = [g_x^{L_m} \ g_y^{L_m}]^T = [00]^T$
• for $L = L_m$ down to 0 with step of -1	
○ Location of point u on image I^L :	$u^L = [p_x \ p_y]^T = u/2^L$
○ Derivative of I^L with respect to x :	$I_x(x, y) = \frac{I^L(x+1, y) - I^L(x-1, y)}{2}$
○ Derivative of I^L with respect to y :	$I_y(x, y) = \frac{I^L(x, y+1) - I^L(x, y-1)}{2}$
○ Spatial gradient matrix:	
	$G = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2(x, y) & I_x(x, y)I_y(x, y) \\ I_x(x, y)I_y(x, y) & I_y^2(x, y) \end{bmatrix}$
○ Initialization of iterative L-K:	$\bar{v}^0 = [00]^T$
○ for $k = 1$ to K with step of 1 (or until $\ \bar{\eta}^k\ < \text{accuracy threshold}$)	
▪ Image difference:	$\delta I_k(x, y) = I^L(x, y) - J^L(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1})$
▪ Image mismatch vector:	$\bar{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x, y)I_x(x, y) \\ \delta I_k(x, y)I_y(x, y) \end{bmatrix}$
▪ Optical flow (Lucas-Kanade):	$\bar{\eta}^k = G^{-1}\bar{b}_k$
▪ Guess for next iteration:	$\bar{v}^k = \bar{v}^{k-1} + \bar{\eta}^k$
○ end of for-loop on k	
○ Final optical flow at level L :	$d^L = \bar{v}^K$
○ Guess for next level $L-1$:	$g^{L-1} = [g_x^{L-1} \ g_y^{L-1}]^T = 2(g^L + d^L)$
• end of for-loop on L	
• Final optical flow vector:	$d = g^0 + d^0$
• Location of point on J :	$v = u + d$

Table 9 - Outline of the LK Tracker [24]

The LK Tracker uses calculations and approaches that are very similar to the ones used by the Harris corner detector. The notation for the image pyramid in Table 9 is that I^L represents the original image at level L in the pyramid. J is used to denote the next image frame. Notice that matrix G is very similar to the gradient used in the Harris

corner detector. The outline in Table 10 shows the pseudo-code for sub-pixel calculations. The approach uses eigenvalues, much like the Harris corner detector does. Using the eigenvalues and distances between pairs of pixels, sub-pixel accuracy is achieved.

<p>Sub-pixel Calculation:</p> <ol style="list-style-type: none">1. Compute the G matrix and its minimum eigenvalue λ_m at every pixel in the image I2. Call λ_{max} the maximum value of λ_m over the whole image3. Retain the image pixels that have a λ_m value larger than a percentage of λ_{max}. This percentage can be 10% or 5%4. From those pixels, retain the local maxima pixels (a pixel is kept if its λ_m value is larger than that of any other pixel in its 3x3 neighborhood)5. Keep the subset of those pixels that make the minimum distance between any pair of pixels larger than a given threshold distance (e.g. 10 of 5 pixels)
--

Table 10 - Pseudo-code for sub-pixel calculation [24]

2.5. Literature Review Remarks

This chapter covered several background areas pertinent to this dissertation. This section covered various types of Kalman Filters, Information Filters, and Particle Filters. The original work presented in the following sections use a combination of background material from the Kalman Filter and Information Filter. Coverage and path planning are not covered in this dissertation. These topics are presented to complete all aspects of SLAM. Various forms of SLAM shown in this chapter cover the general array of SLAM types. They are presented to give contrast to the original SLAM methods presented in the following chapters. Graph SLAM and sparsification are included as key inspirational

components to the original work presented in this dissertation. The biologically inspired section on inertial and vision fusion leads to the use of inertial, encoder, and vision based sensors in this dissertation. It provides a common sense background to why these sensors were selected. The image processing techniques presented in this chapter are used as background material for the next chapter, as new methods were created to solve problems with stereo image processing.

Overall, this literature review provides a brief overview of multi-robot SLAM, from the inspirational end of biology to the full example of SLAM techniques from other researchers. The literature review provides a background to the material presented, contrasting techniques, inspirational techniques, and common algorithms that are referenced throughout the remainder of this dissertation.

CHAPTER 3: LK-SURF

There are two major problems when tracking stereo images over time: correspondence between stereo images, and correspondence over time. Stereo correspondence is where an algorithm must select a pixel or feature in one image and find the same pixel or feature in the other stereo image. This is a very basic, but difficult problem. The next problem is corresponding pixels or features over time. Simply determining where a pixel or feature moved over time.

Lucas-Kanade with Speeded-Up Robust Features (LK-SURF) is the proposed solution to the stereo correspondence over time problem. It uses newer feature extraction techniques to find features in each image. Unlike Harris Corners, these features have descriptors, making matching a much easier problem. The new descriptors are difficult to correspond exactly as they are not points, but small regions of the image. LK-SURF uses a sub-pixel routine to find the most prominent corner inside the small region of the feature. Using the most prominent corner allows temporal tracking of a feature using the LK tracker. At the same time, the most prominent corner provides sub-pixel accuracy on triangulation for better ranging. Using the descriptor of each feature, stereo correspondence becomes more robust than Harris Corner based methods or dense stereo methods, which pick the closest point.

3.1. Stereo Correspondence

Temporal tracking and stereo correspondence have been solved individually. Stereo correspondence can be done with brute force methods that compare patches of pixels until a maximum correlation is found. These methods compute disparities, and the disparity is used to determine range from the camera. These methods are very time consuming and are near impossible to run in real time. These methods do not lend themselves well to tracking features over time since they provide no descriptors or points to track. Stereo correspondence has been done with SIFT, SURF, and other feature extractors, where features are selected from each image, and then correlated using feature descriptors. This approach is generally much faster than the per-pixel or dense disparity approaches, but only contains a sparse set of data. Dense disparity can be accelerated with hardware, however, the resolution and accuracy is not as easily obtained as with sparse features. Looking at Figure 32, the depth is estimated per pixel (represented as a grayscale value). With this sample, it is worth noting the lack of resolution, the missing sections of the lamp, and the entire bookshelf is valued at the same distance. This type of stereo correspondence does not work well for large disparities, or large distances; the sub-pixel accuracy is not feasible because correspondence is done per pixel to match another pixel.

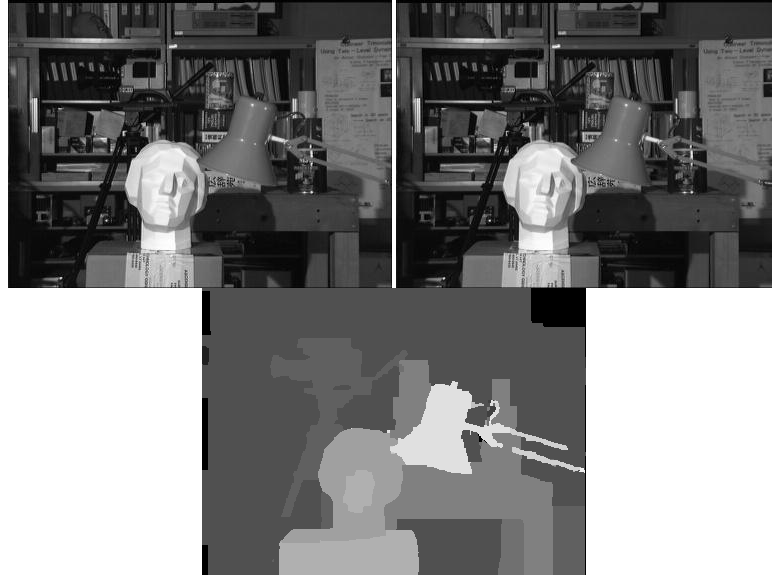


Figure 32 - Example of dense stereo with original left and right image above [22].

3.2. Correspondence Over Time

Correspondence over time can be done in several ways. The most common way is to use a Lucas-Kanade tracker, as mentioned in the previous section. Other ways include matching features over time using SIFT and SURF. The major drawback to using SIFT or SURF over time is that the exact same feature is not guaranteed to be found, and if it is found, there is no guarantee on accuracy. These feature extractors were designed for object recognition, not for optic flow, or precise tracking. Figure 33 shows an example of optic flow using SURF. The blue lines connect the position of matched features between the last and current image. Ideally, the lines should all be small and in the same general direction. In reality, there are many incorrect matches, and even matches that seem correct do not all have the same direction.



Figure 33 - Screenshot of using SURF for optic flow

3.3. Proposed Solution

The natural solution to this problem is to combine the most efficient and reliable methods into a single solution for tracking stereo features over time. Initially features are created by extracting SURF features from the left and right camera image. These features are validated using epipolar geometry determined during camera calibration. Each feature is then placed into a Lucas-Kanade feature tracker with inverted pyramids and tracked frame-to-frame over time. The Lucas-Kanade tracker is used on both the left and right camera frames. There is no guarantee that selected features from SURF will be unique enough to track. To correct this problem, the most prominent corner bounded by the feature is discovered using a sub-pixel corner finding routine (see Table 10). This sub-pixel routine is the same standard techniques as found with the Lucas-Kanade optic flow algorithm. The sub-pixel refinement attempts to find a corner or saddle point in the image, thus increasing the reliability of the feature making it easier to track. If the point is not found, the feature is discarded. Overall, features are tracked in both left and right

camera frames, refined, and validated with epipolar geometry to determine if they are still candidate features for tracking. Table 11 shows the algorithm for LK-SURF.

<p>SURF Feature Initialization</p> <ul style="list-style-type: none"> • Extract <i>left features</i> using SURF • Extract <i>right features</i> using SURF • Refine <i>left feature</i> locations to sub-pixel accuracy • Refine <i>right feature</i> locations to sub-pixel accuracy • Match <i>stereo features</i> using SURF, filter features with a matching strength criteria • Validate <i>stereo match</i> using epipolar geometry • Return all validated <i>stereo matches</i> <p>Stereo LK Feature Tracking</p> <ul style="list-style-type: none"> • Initialize left Lucas-Kanade tracker with all known <i>left features</i> • Initialize right Lucas-Kanade tracker with all known <i>right features</i> • Track <i>left features</i> on the new <i>left image</i> • Track <i>right features</i> on the new <i>right image</i> • Validate newly tracked <i>stereo features</i> using epipolar geometry • Return all validated and updated <i>stereo matches</i> <p>LK-SURF</p> <ul style="list-style-type: none"> • Get input of latest stereo images • Update <i>current features</i> using Stereo LK Feature Tracking • If there are too few <i>current features</i>, or the <i>current features</i> are not covering all four quadrants of the image then <ul style="list-style-type: none"> ○ Get <i>new features</i> using SURF Feature Initialization ○ Append <i>new features</i> to the list of <i>current features</i> • Return <i>current features</i>
--

Table 11 - Outline of LK-SURF algorithm

Thus, LK-SURF combines two well-known algorithms into a single useful solution. Only a few other paper mentions combining SURF with a Lucas Kanade tracker [71] [72], however, these researchers only use this combination for a single camera and track a face or a person. The method was not used for stereo correspondence or for robotics. Figure 34 shows a sample screenshot of two stereo image pairs in sequence with matched features. The newer image on the bottom retains many of the same features from the top image, demonstrating the temporal tracking of spatial features by LK-SURF.

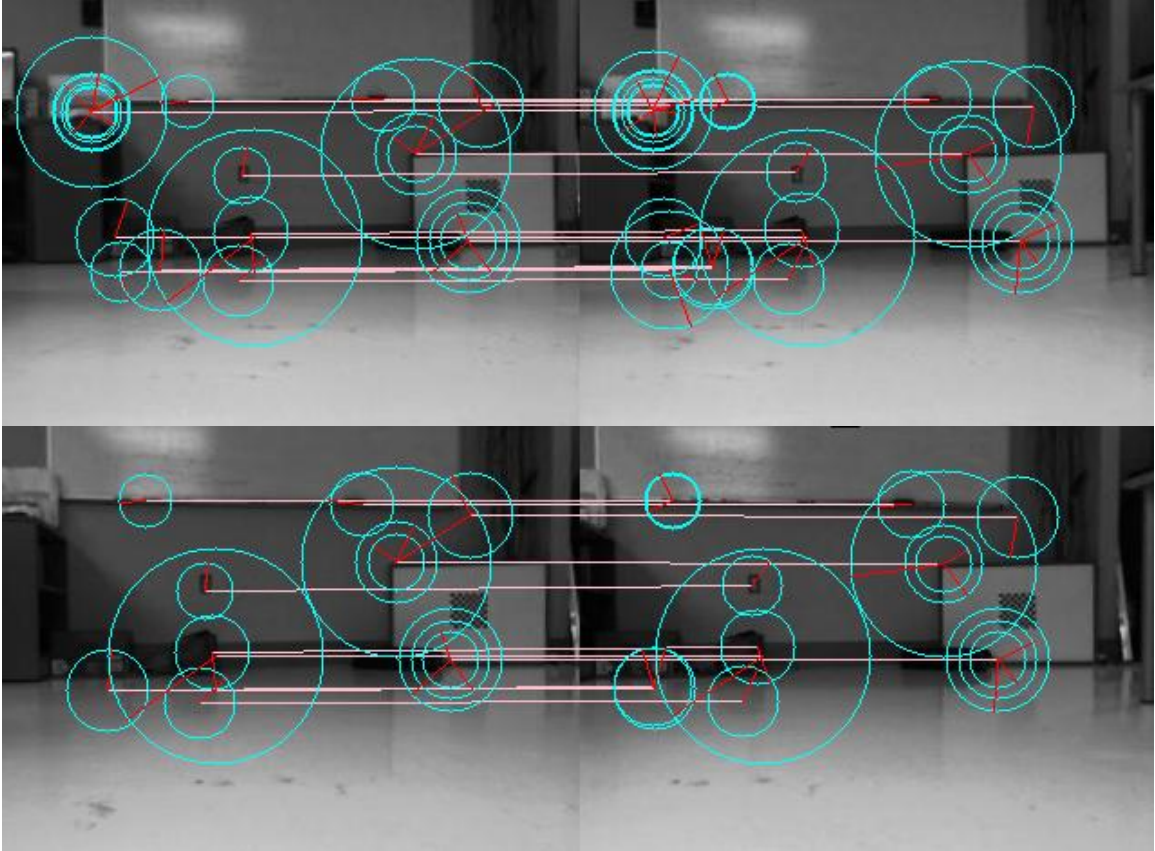


Figure 34 - Sample screenshot of LK-SURF tracking features; older frame on top, recent frame with tracked features on bottom

3.4. Advantages of LK-SURF

LK-SURF allows complete 3D tracking of features over time. A feature has a known range and bearing from the stereo camera rig, and it can be tracked over time, giving us a full 6 degree-of-freedom estimate using egomotion. Figure 35 shows a comparison of a sample path made using SURF and LK-SURF. Both algorithms produce stereo features over time. These features over time are used to calculate the full egomotion the cameras underwent to produce the travelled path. The exact same procedure, match rejection, and data was applied with SURF and LK-SURF, yet the

3.5. Disadvantages of LK-SURF

There are two main disadvantages of LK-SURF. First, SURF stereo matching is not guaranteed to correlate stereo features correctly—this is true of SURF and other descriptor methods as well. Even though epipolar geometry is used to validate stereo matches, it is possible, especially with repeating patterns, to have an incorrect match. Other depth sensors such as LIDAR or SONAR do not suffer from this correspondence issue and can give better depth estimates to features.

The other disadvantage is that the Lucas-Kanade feature tracker was not designed for SURF features, it was designed for corners, and specifically corners detected using a Harris Corner Detector (as mentioned in the previous chapter). Sub-pixel refinement is used to find the most prominent corner within the SURF feature, but is not guaranteed to find a corner that is suitable for the LK tracker. This disadvantage can result in the LK tracker incorrectly tracking SURF features; in particular, if the SURF feature is large and its center does not have any properties resembling a corner, making it a bland or uninteresting feature. This type of feature cannot be tracked well and will usually move incorrectly or not move at all. Epipolar geometry is used to validate against this if the left and right features do not move in the same way, however this might not always filter out features. Figure 36 has a top-down view of features projected into 3D away from the camera shown on top and the source images on the bottom. The picture shows a collection of features, circled in red, that were not correctly tracked. These incorrectly tracked features usually move in a completely erratic direction compared to the other features in the image.

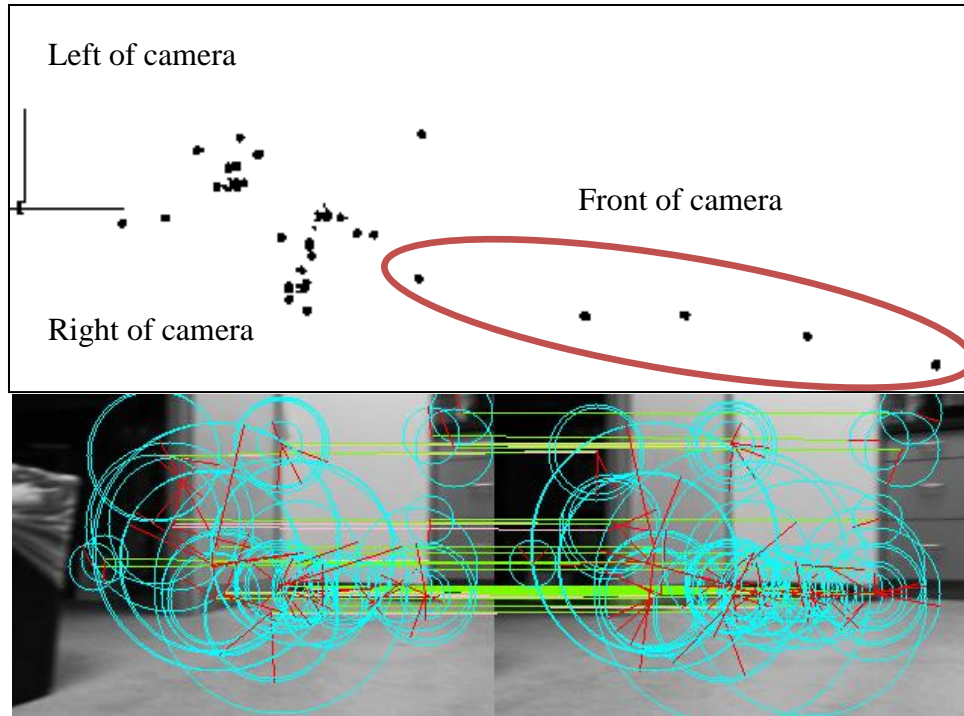


Figure 36 - Example of bland feature tracking. Top: features plotted as a distance away from the camera, stray point circled in red. Bottom: Original image with matched features, excess features tracked in the middle

3.6. Applications

LK-SURF was created to allow 3D ranging to environmental features and to solve the correspondence problem in an efficient manner. Knowing that a stereo video stream is being taken, stereo features can be correlated over time, which acquires an accurate estimate of the same 3D feature over time. This algorithm allows for accurate egomotion calculations to occur in 3D, and eliminate the entire correspondence issue in SLAM. This algorithm was developed in response to the poor performance of feature matching over time with SURF and the errors the invalid matches caused.

CHAPTER 4: CUMRF

The Consolidated Unscented Mixed Recursive Filter (CUMRF) is a newly created filter. It theoretically allows for non-Gaussian noise with nonlinear systems. Using the Gaussian Mixture Model to represent non-Gaussian noise, and modification of the Unscented Kalman Filter (UKF) to handle the mixture model, the new filter showed improvement upon both the EKF and UKF. In the case of a linear system with non-Gaussian noise, the CUMRF performs better than the Kalman Filter does with assumed Gaussian noise.

The main problem with using a Gaussian Mixture Model is the number of Gaussian distributions used to create the mixture model. The number of distributions grows exponentially per iteration, quickly making the system unmanageable. Adding consolidation stages throughout the filter allows the system to remain manageable. The consolidation stage is a step that reduces the number of Gaussian distributions in the Gaussian Mixture Model while maintaining the same total mean and covariance of the mixture.

To address nonlinear dynamics, the UKF is taken as a model. This filter has shown to have better nonlinear performance in estimating the mean and covariance of the system distribution than the EKF [27]. Figure 37 shows a simple example of Gaussian noise propagating through a nonlinear function. The true values are measured through a particle filter, where an extremely high number of particles are used. In both cases, the

UKF performs better than the EKF. The UKF yields results comparable to a third-order Taylor series expansion of the state-model, while Extended Kalman Filters are only accurate to a first-order linearization [45].

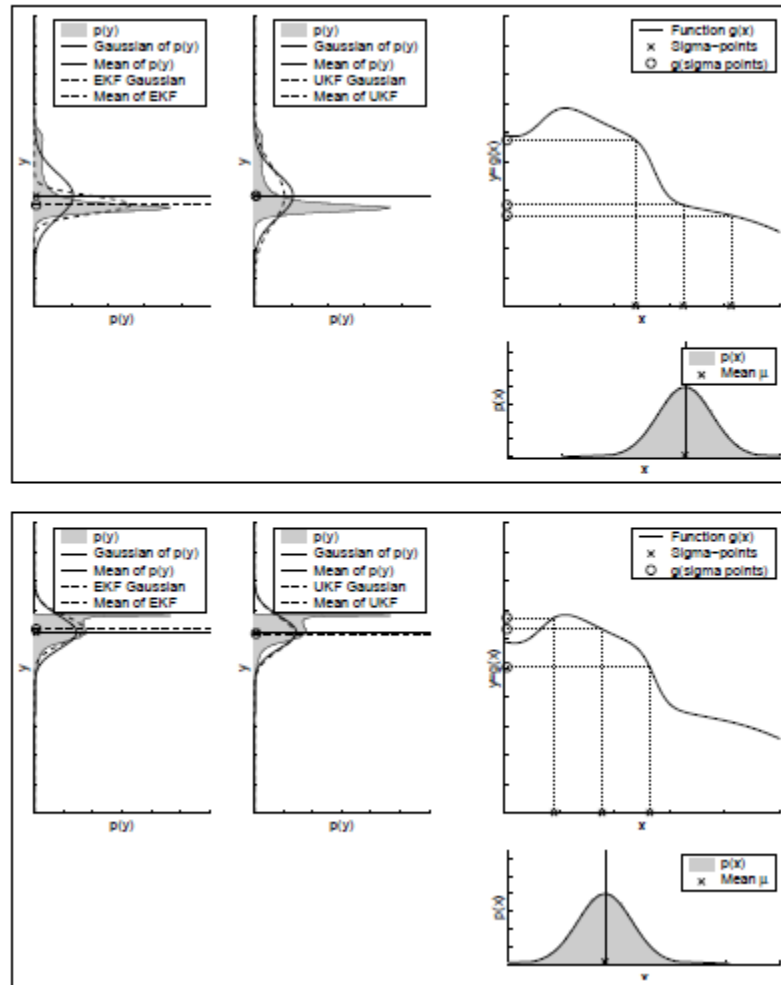


Figure 37 - Comparison of EKF and UKF [27]

4.1. Gaussian Mixture Models

Mathematically, using non-Gaussian noise is usually not advantageous as Gaussian models have nice mathematical properties under linear algebra; most other distributions do not share this property. Adding Gaussians or convolving Gaussians result in Gaussians, making the filtered results manageable. Most researchers simply assume the noise to be Gaussian or use a Particle Filter. The issue with assuming the noise to be Gaussian, when it is truly non-Gaussian, is that you can have situations where the approximated Gaussian is the wrong answer. Example, if there was some finite probability that a robot could be at each of two possible landmarks and the probability distribution around each landmark was with some estimated Gaussian. The probability space is now clearly divided into two distinct Gaussians. Averaging the entire probability space into a single Gaussian would lead to an incorrect estimate and predict that the most likely robot location is between the two landmarks. However, it is at either one or the other, not in the middle. It might be true that the average is between the two, but by only storing the average, we lose the fact that the robot is not at the average of the two locations.

Aside from contrived examples of non-Gaussian situations, there are many real-life examples. For instance, a rangefinder, like a sonar or LIDAR, can only yield positive distances. A sample error distribution for a range sensor would typically follow a Gamma distribution as shown in Figure 38. This example actually has the same mean and variance for both the Gamma distribution and Gaussian distribution, but note that the Gaussian distribution includes negative values as a possibility. This very nature indicates the use of a Gaussian is inappropriate.

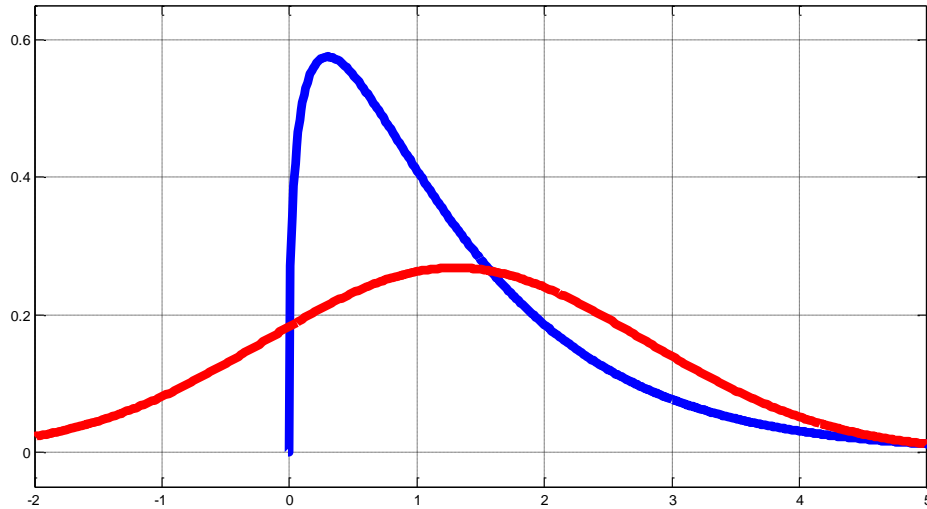


Figure 38 - Comparison of Gamma and Gaussian distribution

A solution to the problem in Figure 38, where values should only be positive and non-symmetrical, would be a Gaussian Mixture Model. A Gaussian Mixture Model example is shown in Figure 39, where two Gaussians are added together in a 60% and 40% ratio creating the green non-Gaussian distribution. This combination can already begin resembling unbalanced distributions and can approximate almost any distribution much like a Fourier series, or a wavelet representation using Gaussians.

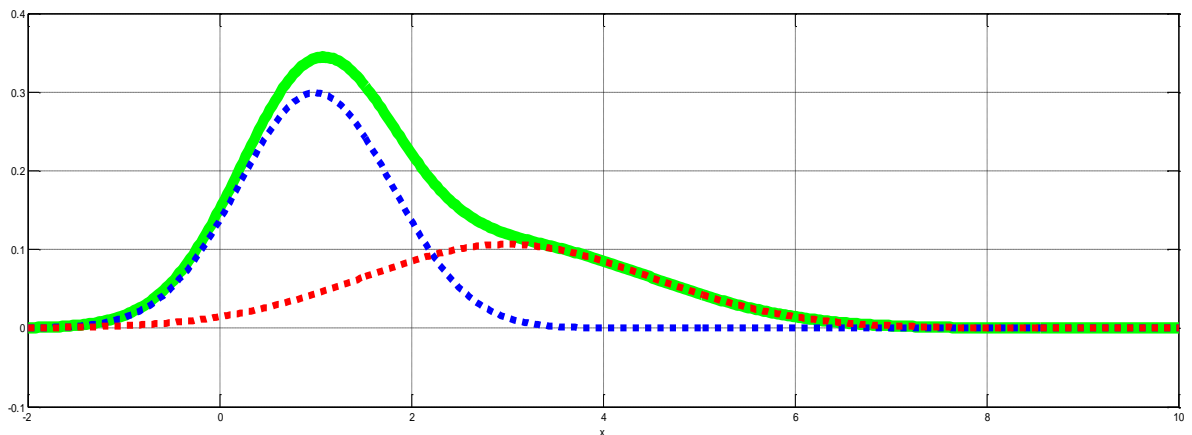


Figure 39 - Sample Gaussian mixture

4.2. Merging Sigma Points

The consolidation stage is one of the most important parts of this filter. Consolidation takes two Gaussians, and approximates the weighted average Gaussian of the two, maintaining the same total system mean and covariance. At a worst-case scenario, merging every Gaussian in the mixture model into a single Gaussian would approach a normal Kalman-like filter with a single Gaussian. An efficient merging process is very important in CUMRF. This is due to the exponential increase in the number of Gaussians used in the mixture model per filter iteration. The details of this problem are explained in detail in the next section. The primary concern of this section is how Gaussians are merged, and how merging is completed efficiently.

The most efficient way to merge Gaussians, when using the Unscented Kalman Filter, is to reuse the sigma points in the Unscented Transform [49], requiring no additional calculations, matrix math, or sampling. Similar to the UKF algorithm as outlined in Table 4, the mean is calculated using the same transform calculation, but given two sets of sigma points with weights. The covariance matrix is calculated in a similar fashion, but slightly differing where different coefficients are used, both center sigma points and calculated mean are used as well. Table 12 shows the complete algorithm for merging Gaussians using the sigma points from the Unscented Transform, producing a single Gaussian.

Calculate Merged Gaussian ($\Sigma_1, w_1, \Sigma_2, w_2, L, \alpha, \beta, \kappa$)

Where Σ represents the set of sigma points, w is the weight of the set, L the dimension of the system, and α , β , and κ as parameters of the Unscented Transform

$$\lambda = \alpha^2(L + \kappa) - L$$

$$w_1^{norm} = w_1 / (w_1 + w_2)$$

$$w_2^{norm} = w_2 / (w_1 + w_2)$$

$$X_{mean} = \left(\begin{array}{l} w_1^{norm} \frac{\lambda}{L + \lambda} \Sigma_1^1 + w_2^{norm} \frac{\lambda}{L + \lambda} \Sigma_2^1 \\ + \sum_{i=2}^{2*L+1} \left[\frac{w_1^{norm}}{2(L + \lambda)} \Sigma_1^i + \frac{w_2^{norm}}{2(L + \lambda)} \Sigma_2^i \right] \end{array} \right)$$

$$P_{covariance} = \sum_{i=2}^{2*L+1} \left(\begin{array}{l} \frac{w_1^{norm}}{2L} \left(\sqrt{\frac{L}{L + \lambda}} (\Sigma_1^i - \Sigma_1^1) + \Sigma_1^1 - X_{mean} \right) \\ * \left(\sqrt{\frac{L}{L + \lambda}} (\Sigma_1^i - \Sigma_1^1) + \Sigma_1^1 - X_{mean} \right)^T \\ + \frac{w_2^{norm}}{2L} * \left(\sqrt{\frac{L}{L + \lambda}} * (\Sigma_2^i - \Sigma_2^1) + \Sigma_2^1 - X_{mean} \right) \\ * \left(\sqrt{\frac{L}{L + \lambda}} * (\Sigma_2^i - \Sigma_2^1) + \Sigma_2^1 - X_{mean} \right)^T \end{array} \right)$$

Table 12 - Algorithm for merging Gaussians using sigma points

4.3. Filter Details

The filter is designed to take in any Gaussian Mixture Model for the previous estimated state covariance, the predictive state noise, and the measurement noise. In addition, since each distribution is represented as a Gaussian Mixture Model, there is also a weight per Gaussian, and a mean per Gaussian. Thus, this system automatically handles nonzero Gaussian noise, which would require a trivial coordinate change in the normal case.

Figure 40 shows a naive approach at producing a Kalman-like recursive filter with Gaussian Mixture Models, represented by a number of Gaussians. The prediction, observation, and final estimate stages can use the Unscented Transform from the UKF, but the diagram works for an EKF based filter as well. As the diagram shows, the final state estimate will yield a sizable number of Gaussians, which will be returned to the beginning of the filter in the next iteration causing an exponential growth in the number of Gaussians. This approach is simply not acceptable.

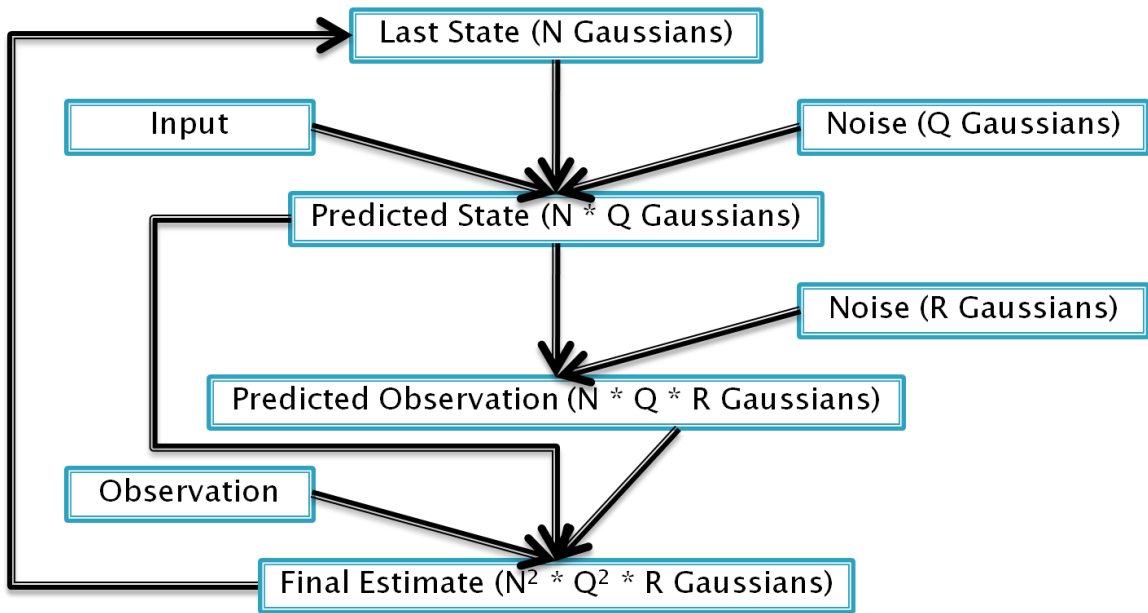


Figure 40 - Naive Multi-Gaussian recursive filter

Figure 41 shows a modified version of the naive approach. Simply by adding stages to consolidate the total number of Gaussians, the exponential growth can be forcibly capped. By consolidating the size of the Gaussian Mixture Model at each stage using the approach outlined in the previous section, the system maintains a manageable set of Gaussians instead of an exponentially growing set of Gaussians.

The consolidation stage applies a simple heuristic similar to a particle filter weighting to find which two Gaussians are closest to each other. This method can be used with any heuristic desired. The tested heuristic was the summation of the difference in mean over each direction of the covariance matrix in the direction of the other mean plus ratio of the relative sizes of the covariance matrices. This heuristic attempts to merge Gaussians that are close to each other with similarly sized covariance matrices.

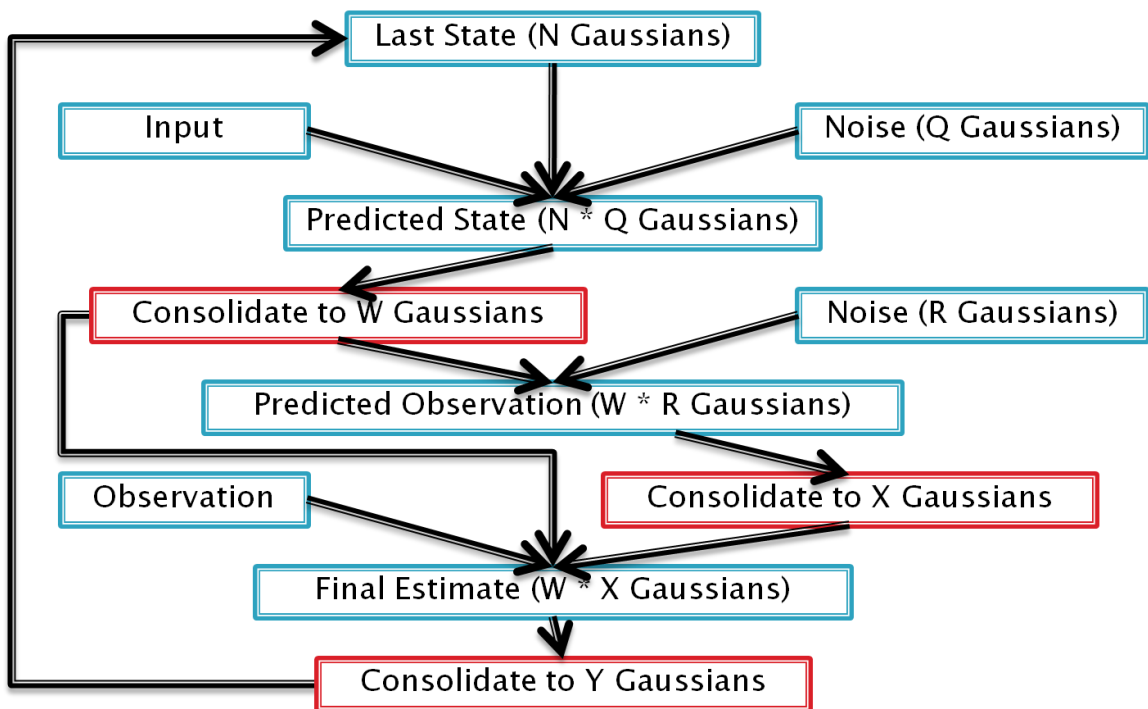


Figure 41 - Consolidated approach to Multi-Gaussian recursive filtering

A further note about this filter, if the number of Gaussians is capped to only a single Gaussian, then we have a normal Kalman-like Filter. Furthermore, the Unscented Kalman Filter will yield the optimal results if the dynamics are linear functions. If the number of Gaussians allowed in each Gaussian Mixture Model after consolidation is quite large, then the filter resembles a Particle Filter, with many sigma points sampling the probability space. Unlike the Particle Filter, this technique yields the same results

each time, no random selection or random sample consensus (RANSAC) approaches are taken, making this a deterministic system. The Particle Filter uses random selection of states in estimated distributions; it is never guaranteed to yield optimal results.

4.4. Example

A key example to illustrate the power of this new filter is a direct comparison to the optimal filter of the linear Kalman Filter with a linear system and only two Gaussians that are nearby, still yielding an overall zero-mean noise. Table 13 shows the exact equations used in this example. These particular equations were chosen simply to illustrate how well the filter can work, and give a system with some interesting dynamics.

<ul style="list-style-type: none"> • $A = [0 \ -0.9; \ -0.9 \ 0]$ • $B = [1; \ -1]$ • $H = [1 \ 0]$ • $U_k = 100 * \sin(k/1000 * 50 * \pi) / \sqrt{k}$ <ul style="list-style-type: none"> ○ This just looks nice, no particular reason why this formula • $Q = 25\% \ Q_a \text{ and } 75\% \ Q_b$ <ul style="list-style-type: none"> ○ $\text{Mean}(Q_a) = [0; \ 1.5], \text{Var}(Q_a) = [0.1 \ -0.0625; \ 0.125 \ 0.25]$ ○ $\text{Mean}(Q_b) = [0; \ -0.5], \text{Var}(Q_b) = [0.05 \ 0.025; \ -0.03625 \ 0.075]$ • $R = 80\% \ R_a \ 20\% \ R_b$ <ul style="list-style-type: none"> ○ $\text{Mean}(R_a) = 0.025, \text{Var}(R_a) = 1$ ○ $\text{Mean}(R_b) = -1, \text{Var}(R_b) = 2$ • $X_0 = [1 \ 1];$ • $P_0 = \text{Var}(Q)$
--

Table 13 - Example equations

The next few figures (Figure 42, Figure 43, Figure 44, and Figure 45) show plots of the perfect sensor data, noise added on top of the sensor data, the Kalman estimate

overlaid on the noise, the CUMRF estimate overlaid on the noise, and a comparison of the estimate error of the two filters.

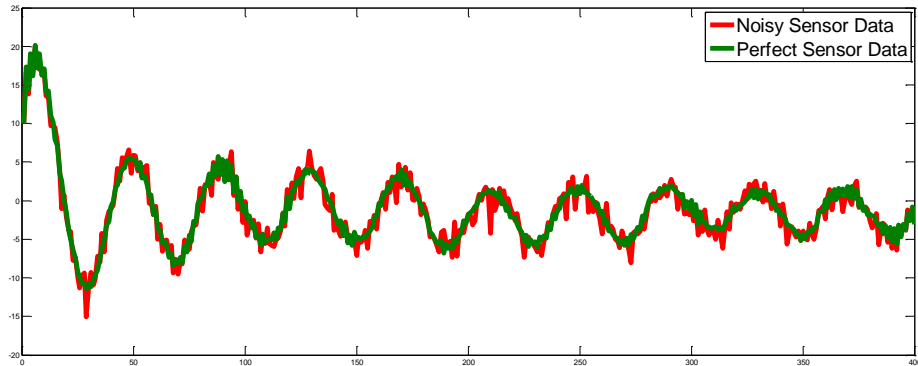


Figure 42 - Example sensor noise sampled over time

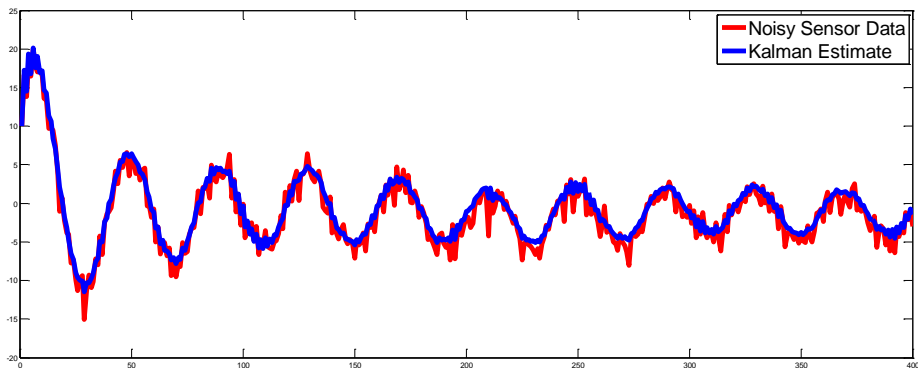


Figure 43 - Kalman estimate and sensor input data

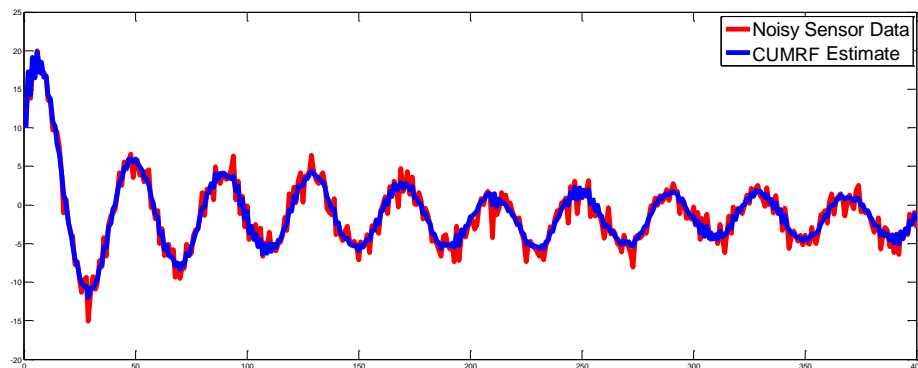


Figure 44 - CUMRF estimate and sensor input data

At first glance, Figure 43 and Figure 44 look almost the same; however, there are subtle differences. The Kalman estimate is actually not always centered on the data as

often as the CUMRF estimate. Figure 45 shows the absolute difference between the true data and filtered data for the two filter estimates. In this example, the CUMRF clearly performs better due to a better representation of the system noise.

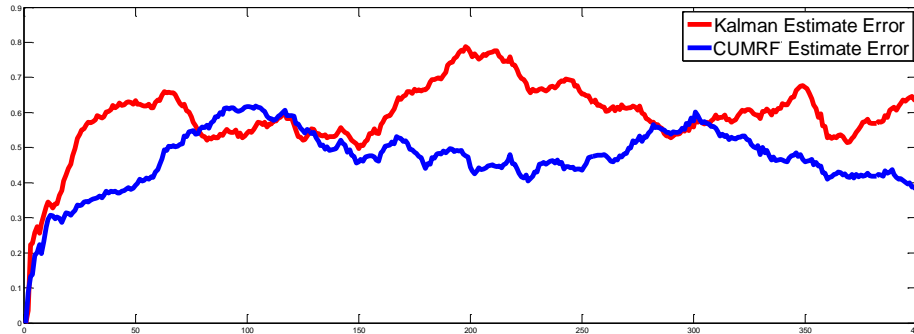


Figure 45 - Comparison of Kalman and CUMRF estimate error

4.5. Applications

The Consolidated Unscented Mixed Recursive Filter was originally designed to take the place between the EKF SLAM and FastSLAM. Where FastSLAM still suffers from a large number of particles and updating a large amount of Extended Kalman Filters, and where EKF SLAM suffers from the lack of flexibility for non-Gaussian systems and does not always deal well for with nonlinear systems. The downside is that the CUMRF is a factor more computationally expensive than the EKF or UKF based on the maximum allowed Gaussians in the mixture model. It can be as simple and efficient as the EKF or UKF or as complex and accurate as the Particle Filter.

For practical purposes, the entire CUMRF is not used in the final solution of this dissertation, but the Gaussian merging, and unscented transform propagations are used

for sensor modeling (see Chapter 10: Calibration and Modeling), and other filters in the algorithm.

The main use of CUMRF in this dissertation is the modeling of the error in the optical system. The resulting noise on the optical system ends up being non-Gaussian. A Gaussian mixture is able to model the error and show statistical significance (see Chapter 10: Calibration and Modeling). Another use of CUMRF is the actual robot model. The model can be made using a neural net, which does not have an easy to calculate Jacobian, making the Extended Kalman Filter difficult to use. If the robot model yields non-Gaussian noise, CUMRF can be used in the prediction stage of Forgetful SLAM (see Section 6.4: Modification for Neural Net Based Models); however, the resulting Gaussian Mixture Model is consolidated to a single Gaussian in order to reduce computational complexity of the SLAM routines.

CHAPTER 5: ROBUST KALMAN FILTER

The Kalman Filter is the optimal solution to finding the best estimate of the state of a linear dynamics system with independent Gaussian noise. A practical problem with using the Kalman Filter is that it is designed for Gaussian noise. When unmodeled outliers are observed, the Kalman Filter does not yield expected results. This attribute makes the Particle Filter more desirable in robotics; however, what if the Kalman Filter could be more robust to outliers? The Robust Kalman Filter works with systems that have many independent observations at once, as with LIDAR range finders, or in the case of this dissertation, a collection of points found in 3D through cameras. Typically, robust filters simply remove outliers before entering the Kalman Filter stage, but this is not always feasible, nor is it reliable given that predictions and deviations from expectations need to be considered.

A commonly used outlier rejection mechanism in statistics is the X84 outlier rejection rule [73]. The X84 outlier rejection rule uses the median and median absolute deviation (MAD) to remove outliers instead of the traditional mean and standard deviation of a series of data. The X84 rule works with single dimension statistics, outliers are removed if they are more than a certain number of MADs away from the median. The X84 rule is used in image processing for 2D feature tracking to remove outliers that do not agree with the general affine transformation of the image [74]. The

same idea is extended to the Kalman Filter, but instead of 1D or 2D, the filter works with any number of dimensions.

This dissertation requires robust filtering of 3D image features and the effect each observation has on a robot pose of seven dimensions, which is not covered by the X84 rule. A common mechanism in statistics and computer vision is Principal Component Analysis (PCA), which can be used to transform high dimensional states into low order spaces[75]. PCA works well in reducing high dimensional states into low spaces as shown in the Eigenface algorithm for face recognition, which uses PCA to reduce images into low orders spaces [76]. Data projected along the first principal component has the highest variance in a single dimension, thus change in robot pose caused by an observation is projected to a single dimension using PCA [75].

The Robust Kalman Filter removes observations between the prediction and update stage of the Kalman Filter. After the system is predicted, observations are predicted. The Kalman gain associated with each observation is estimated separately and used to determine how much each observation changes the state. Each observation generates a vector in the same dimension as the robot pose, which indicates how much the single observation affects the state. PCA is used to map the vectors into a single dimension. The X84 rule is applied on the mapped vectors, removing all observations that are beyond a certain number of MADs away from the median. The remaining observations are applied to the state using the normal Kalman Filter update stage.

5.1. Removing Outliers from the Kalman Filter

The first step in removing outliers is finding a way to map the high-dimensional observations into a one-dimensional space. One area of inspiration comes from FastSLAM, where particles in FastSLAM are weighted based on observations matching expectations [27]. With FastSLAM, each observation is given a weight based on the expected observation, the actual observation, and the covariance of the observation. The weights in FastSLAM assume perfect state knowledge to produce a weight. The Robust Kalman Filter uses a different weighting metric; the weight is derived from the observation covariance and expectation covariance since a perfect state is not assumed. The Robust Kalman Filter uses the actual effect the observation has on the state if the optimal Kalman gain is applied from the difference between the observation and predicted observation. By using the change in the state as a common metric, many observations of different dimensions can be compared. Instead of comparing just the magnitude of error from the prediction, a more accurate mapping uses PCA to produce the largest variation amongst the mapped data [75]. In the simplest form, the largest eigenvector of the observed changes to the system is the largest principal component. Table 14 shows the equations used to map the observations into a single dimensional weighting, where the weight of each observation is based on a multivariate normal distribution from the predicted observation, multivariate normal distribution from the state prediction, the Kalman gain, and the PCA mapping.

Observation Weighting: Z_i = Observation $Z_i^{\text{Predicted}}$ = Predicted Observation \mathbf{Q}_i = Observation Covariance \mathbf{P} = State Covariance \mathbf{H}_i = Observation Jacobian w_i = Observation Weight

$$\mathbf{K} = \mathbf{P}\mathbf{H}_i^T (\mathbf{H}_i\mathbf{P}\mathbf{H}_i^T + \mathbf{Q}_i)^{-1}$$

$$\mathbf{x}_i = \mathbf{K} (Z_i - Z_i^{\text{Predicted}})$$

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$$

$$\tilde{\mathbf{X}} = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$$

 \mathbf{v} = largest eigenvector by eigenvalue of $(\tilde{\mathbf{X}})$

$$w_i = \mathbf{v}^T \mathbf{x}_i$$

Table 14 - Observation weighting for removal

The X84 outlier rejection rule [73] uses the median as the center of the data instead of using the mean of a set of data. Furthermore, instead of using the standard deviation, use the median absolute deviation (MAD) as shown in Table 15. The use of this rejection rule can be viewed as determining the corrupted Gaussian that has been altered by the addition of outliers [73][74]. The median and MAD are not affected by outliers, unlike the mean and standard deviation. With a Gaussian distribution, 1.4826 MADs is equivalent to a standard deviation. The original X84 rule allows any value of k (the number of MADs away from the median); however, the best value when applied to removing outliers generated by LK-SURF is three. Three MADs are nearly equivalent to two standard deviations under a Gaussian distribution [73]. Under a pure Gaussian

distribution, three MADs cover 95% of the data, which means if the observed features had pure Gaussian distributions, only 5% would incorrectly be removed.

Outlier Removal:

$$mad(X) = \underset{x \in X}{median}(|x - median(X)|)$$

$$lowerBound(X, k) = median(X) - k * mad(X)$$

$$upperBound(X, k) = median(X) + k * mad(X)$$

$$X_{filtered} = \{upperBound(X, 2) \geq x \geq lowerBound(X, 2) \mid x \in X\}$$

Table 15 - Outlier removal for single dimensional arrays

Using PCA to map the change in system state to a single dimensional weighting works well due to the sensitivity of PCA [75]. PCA will give the principal vector that causes the data to have the highest variance in one dimension. This technique is sensitive to outliers, causing outliers to appear at extremes. Combining PCA with the X84 rule provides a robust way to remove outliers. This technique is agnostic to the number of dimensions, number of measurements, and even if the observations are of different dimensions, the weighting is based on the how much the state is affected.

In order to incorporate these weighting and outlier removal functions, the Kalman Filter needs some modifications. After the prediction stage, the next step in the measurement stage is to calculate the residuals. Here the new section is injected, before the residuals are calculated. The weighting function is passed the predicted state, the predicted state covariance, the predicted measurement values, the actual measurement values, the observation H matrix, and the observation covariance R. Assuming that each observed feature is independent from other features, we can extract each feature's covariance from R and each feature's expected value and measured value. Each feature is given a weight according to Table 14, and then the entire set of features is filtered based

on the calculated weight as described in Table 15. Table 16 shows the entire Robust Kalman Filter. A version of the Robust Kalman Filter for nonlinear system dynamics would have the exact same modifications as the Extended Kalman Filter. If there are observations that cannot be placed into the robust observations section, or the implementer simply does not want the observation to be removed, then simple additions are made. An additional observation and update stage are applied after the prediction stage but before the robust removal. This is equivalent to two Kalman Filters operating in series, first the state is updated using sensors that are not to be removed, and then a second robust filter is applied where the system dynamics are the identity function since no time has passed. The filters are applied in this order to ensure that the outlier removal stage has the best estimate for the state prediction and observation prediction.

System Equations:	$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \quad \mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$ $\mathbf{w}_k \sim N(0, \mathbf{Q}_k) \quad \mathbf{v}_k \sim N(0, \mathbf{R}_k)$
Prediction Phase:	
Predicted state:	$\mathbf{x}_{k k-1} = \mathbf{F}_k \mathbf{x}_{k-1 k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}$
Predicted estimate covariance:	$\mathbf{P}_{k k-1} = \mathbf{F}_k \mathbf{P}_{k-1 k-1} \mathbf{F}_k^T + \mathbf{Q}_{k-1}$
Removal Phase:	
Measurement prediction:	$\tilde{\mathbf{z}}_k = \mathbf{H}_k \mathbf{x}_{k k-1}$
Calculate observation weights:	$\left\{ \begin{array}{l} \forall i \in \text{observed features} \left\{ \begin{array}{l} \mathbf{K} = \mathbf{P} \mathbf{H}_i^T (\mathbf{H}_i \mathbf{P} \mathbf{H}_i^T + \mathbf{R}_k)^{-1} \\ \mathbf{x}_i = \mathbf{K} (Z_i - Z_i^{\text{Predicted}}) \end{array} \right. \\ \bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad \tilde{\mathbf{X}} = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \\ \mathbf{v} = \text{largest eigenvector by eigenvalue of } (\tilde{\mathbf{X}}) \\ \forall i \in \text{observed features} \left\{ w_i = \mathbf{v}^T \mathbf{x}_i \end{array} \right.$
Calculate feature to remove:	$\left\{ \begin{array}{l} \text{mad}(X) = \underset{x \in X}{\text{median}}(x - \text{median}(X)) \\ \text{lb}(X, k) = \text{median}(X) - k * \text{mad}(X) \\ \text{ub}(X, k) = \text{median}(X) + k * \text{mad}(X) \\ I_{\text{keep}} = \{i \mid \text{ub}(\mathbf{W}_k, 2) > x > \text{lb}(\mathbf{W}_k, 2), \mathbf{w}_k^i \in \mathbf{W}_k\} \\ \mathbf{M}_k = \text{row reduced identity matrix of} \\ \text{size}(\text{length}(I_{\text{keep}}) * \text{sizeof}(\text{feature}), \text{sizeof}(\mathbf{z}_k)) \end{array} \right.$
Reduce prediction:	$\hat{\tilde{\mathbf{z}}}_k = \mathbf{M}_k \tilde{\mathbf{z}}_k$
Reduce observation:	$\hat{\mathbf{z}}_k = \mathbf{M}_k \mathbf{z}_k$
Reduce observation matrix:	$\hat{\mathbf{H}}_k = \mathbf{M}_k \mathbf{H}_k$
Reduce observation covariance:	$\hat{\mathbf{R}}_k = \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T$
Update Phase:	
Measurement residual:	$\mathbf{y}_k = \hat{\mathbf{z}}_k - \hat{\tilde{\mathbf{z}}}_k$
Residual covariance:	$\mathbf{S}_k = \hat{\mathbf{H}}_k \mathbf{P}_{k k-1} \hat{\mathbf{H}}_k^T + \hat{\mathbf{R}}_k$
Kalman gain:	$\mathbf{K}_k = \mathbf{P}_{k k-1} \hat{\mathbf{H}}_k^T \mathbf{S}_k^{-1}$
Updated state estimate:	$\mathbf{x}_{k k} = \mathbf{x}_{k k-1} + \mathbf{K}_k \mathbf{y}_k$
Updated estimate covariance:	$\mathbf{P}_{k k} = (\mathbf{I} - \mathbf{K}_k \hat{\mathbf{H}}_k) \mathbf{P}_{k k-1}$

Table 16 - Robust Kalman Filter algorithm

5.2. Adaptations of the Robust Kalman Filter

The Robust Kalman Filter can easily be extended to nonlinear systems through the exact same process as the Extended Kalman Filter. Simply obtaining the state matrices through Jacobians will suffice. The Unscented Kalman Filter is a possible alternative, but requires the recalculation of sigma points once observations are removed.

Modifying this filter for observations that are not collections of ranging data or feature positions is possible. Simply expanding the system state to include a time history and expand the measurements to be a time history will allow the filter to perform Robust Kalman Filtering against a time series of data, where particular samples can be removed. This can be forward looking, backward looking, or both to any degree, resembling a median filter.

Observations that the implementer does not wish to include in the Removal Phase should be applied to the system before the Removal Phase. The standard Kalman Filter observation stage and update stage should be applied to the system using the selected observations. This is equivalent to performing a Kalman Filter on system using the selected observations, then performing the Robust Kalman Filter on the system using the identity function as the system dynamics since no time has passed. This gives the system the best estimate of its current state before comparing the removable observations to the system.

5.3. Example

The Robust Kalman Filter was designed for eliminating outliers. The stereo cameras on the test platform (see Chapter 9: Hardware and Design) sense the location of 3D features using LK-SURF (see Chapter 3: LK-SURF). LK-SURF removes many outliers using SURF feature matching techniques, and using epipolar geometry; however, it does not remove all outliers. The Robust Kalman Filter does not suffer as much from poorly matched or tracked features, which drastically alter the estimated position of the robot in the Kalman Filter. Poorly matched or tracked features cannot be modeled as Gaussian noise as they are incorrectly matched in a separate process and are not based on noise, but on shortcomings of real life algorithms. It is possible that the environment is not completely static, which introduces many more errors that are not modeled. So long as 50% of the observed features align with the expected model, the X84 outlier rejection rule should work [74].

Figure 46 shows two paths generated by Forgetful SLAM (see Chapter 6: Forgetful SLAM) using the same data. Forgetful SLAM uses a linearized version of the Robust Kalman Filter. For comparison, the robust filter is replaced with an EKF. The normal Forgetful SLAM routine with the Robust Kalman Filter is shown on the left, while the right has the EKF. The EKF version cannot handle the unexpected observations and produces large errors in the path. The Robust Kalman Filter removes outliers and produces a reasonable path. The Robust Kalman Filter is not perfect; on the right side of the plot, some outliers affect the path. It appears to correct itself, but that is merely luck. The estimated true path is shown on the bottom of the figure as a reference.

The estimate is made using scale corrected wheel encoders and drift corrected gyroscopic angles, such that the path ends in the correct location.

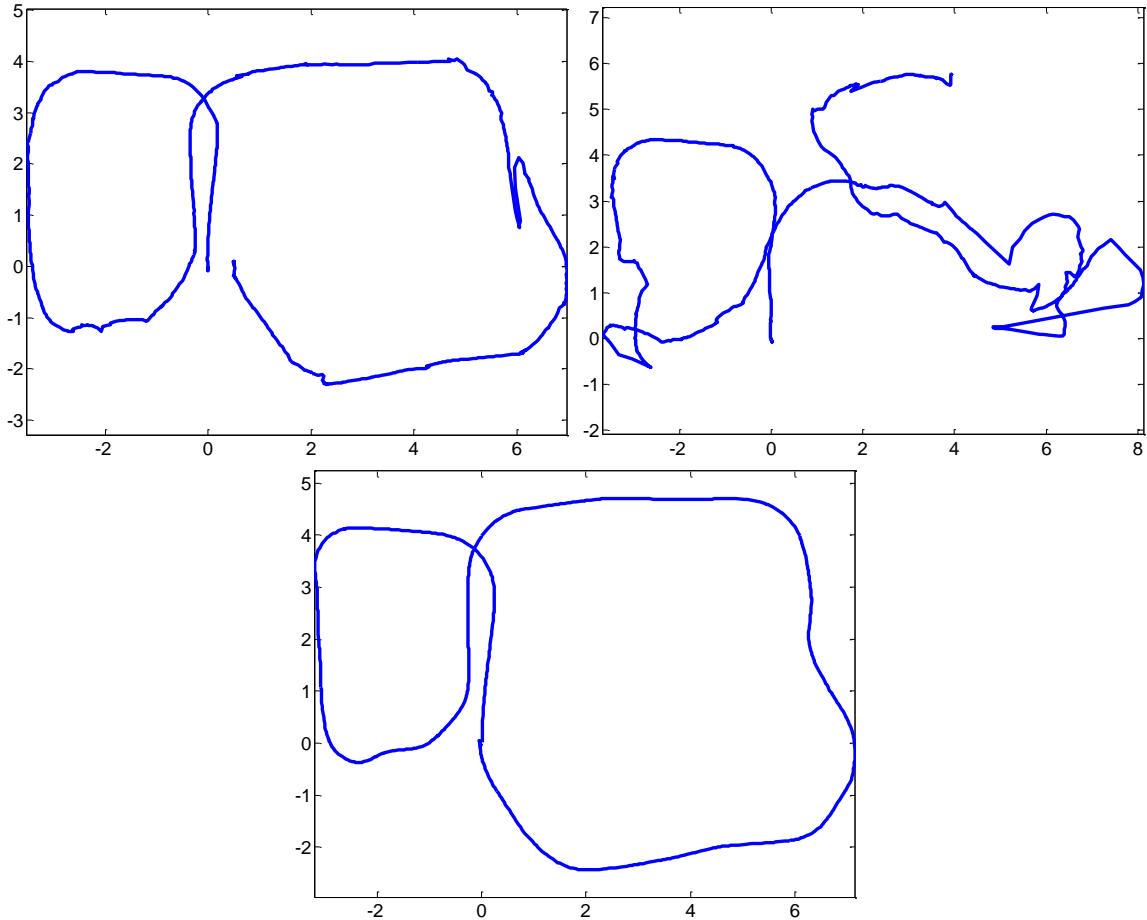


Figure 46 - Comparison of Robust Kalman Filter vs. Extended Kalman Filter.
Left: Robust Kalman Filter path; Right: Extended Kalman Filter path; Bottom: Estimate of true path

5.4. Analysis

In order to determine if the Robust Kalman Filter actually removed outliers, a series of paths were compared. The set of paths are taken over different distances and in different shapes. Comparisons are made between an inertial path based on wheel encoders and gyroscopic angle, Forgetful SLAM using an EKF, and Forgetful SLAM using a linearized version of a Robust Kalman Filter (see Chapter 6: Forgetful SLAM).

The inertial path does not include visual features from LK-SURF; this path is included as a baseline. Each path is given an estimate percent error over distance travelled. The wheel encoders measure the total distance travelled, and the error is determined by how far the end of the path is from the true location. In each of the test paths, the true location of the end of path is the same as the start location.

Path	Wheel Encoder and Gyroscope Path	Forgetful SLAM using an EKF	Forgetful SLAM using Robust Kalman Filter
Short Clockwise Path	0.85%	2.75%	1.06%
Short Counterclockwise Path	2.62%	22.99%	2.59%
Long Clockwise Path	2.44%	2.48%	1.76%
Long Counterclockwise Path	3.65%	24.65%	3.23%
Figure-Eight Path (Figure 46)	2.75%	18.41%	1.35%
Four Loop Path	3.85%	8.50%	2.65%
Unusual Search Path	8.34%	169.83%	3.45%
High Speed Dynamics Test Path	7.34%	10.41%	1.61%

Table 17 - Filter performance comparison using percent error over distance travelled. Eight paths used to compare inertial paths, Robust Kalman Filter paths, and Extended Kalman Filter paths.

Table 17 contains the percent error over distance for various paths and filters. The Robust Kalman Filter usually performed the best. In order to improve the simple

encoder and gyroscope path, the Robust Kalman Filter and EKF must correct small deviations in distance and gyroscopic drift. Given that image features have no drift, and provide another source of data for distance, the Robust Kalman Filter and EKF have the capability to perform better than the simple path if outliers are removed. The EKF always performed worse than the simple path and the Robust Kalman Filter.

When the EKF appears to perform similarly to the Robust Kalman Filter and simple path, based on percent error over distance travelled, the path shape is visually incorrect. Table 18 shows a set of small images of each path. This table is presented to show the various path shapes, and visually how each filter performed on the path. The Robust Kalman Filter consistently matches shape to the simple encoder and gyroscopic path. The EKF path suffers from outliers, causing the path to be distorted.

Given the examples in this section, the Robust Kalman Filter appears to remove most outliers. It is not immune to outliers; some paths show anomalies indicating outliers are present. The outlier rejection rule used only works when outliers make up less than 50% of the samples. There is no guarantee that there are any correctly tracked features from LK-SURF, therefore incorrect features can enter the filter. The Robust Kalman Filter consistently performs better than the EKF based on percent error over distance travelled and heuristically based on path shape.

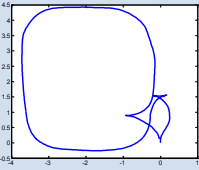
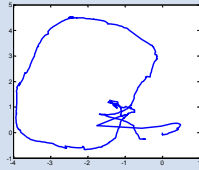
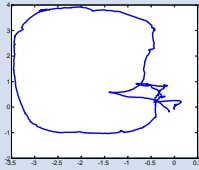
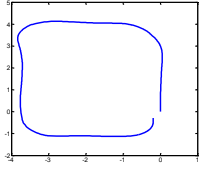
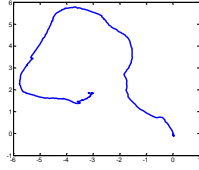
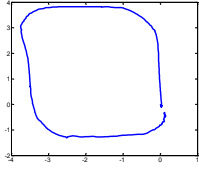
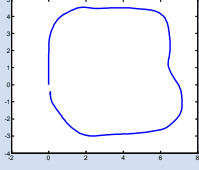
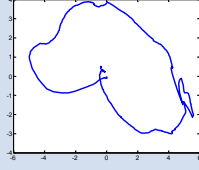
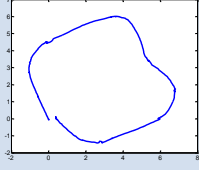
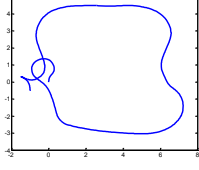
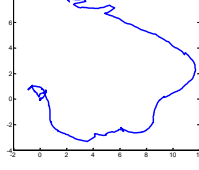
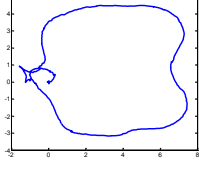
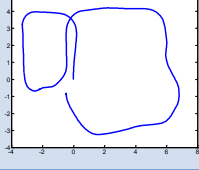
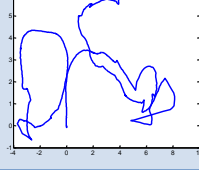
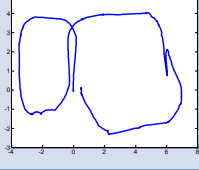
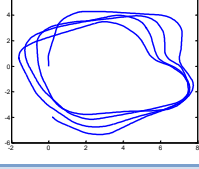
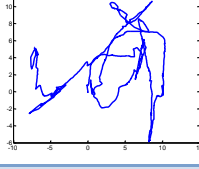
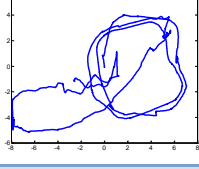
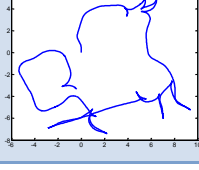
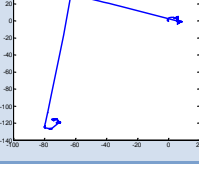
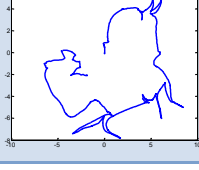
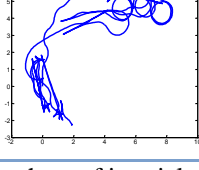
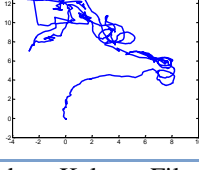
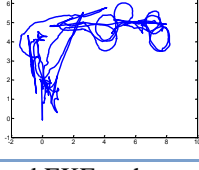
Path	Wheel Encoder and Gyroscope Path	Forgetful SLAM using an EKF	Forgetful SLAM using Robust Kalman Filter
Short Clockwise Path			
Short Counterclockwise Path			
Long Clockwise Path			
Long Counterclockwise Path			
Figure-Eight Path (Figure 46)			
Four Loop Path			
Unusual Search Path			
High Speed Dynamics Test Path			

Table 18 - Plot snapshots of inertial paths, Robust Kalman Filter paths, and EKF paths

5.5. Applications

The Robust Kalman Filter was created in response to the poor performance of the EKF using the features tracked with LK-SURF. Kalman-based filters typically cannot ignore any anomalies, and a simple mean or median filter on the data appear to be superior to the EKF. Seeing this as the major disadvantage of the EKF, the Particle Filter was investigated; however, its implementation was far too computationally expensive, and did not make corrections to the state, the filter simply propagated particles that agreed with measurements.

In this dissertation, the Robust Kalman Filter is used for the low-level sensor fusion, where the random anomalies of the optical system require filtering. The optical system performs stereo feature matching, and is subject to various sources of error. Hardware limitations with the camera create typical Gaussian distributions (see Section 10.2.3: Modeling Error), but in addition to modeled noise, the system has the non-Gaussian errors of selecting features, and matching features. Since the optical system is a major component of the implemented version of Cooperative SLAM in this dissertation, it is important to remove unexpected noise early. The Robust Kalman Filter is directly used in Forgetful SLAM, which is described in the next chapter.

CHAPTER 6: FORGETFUL SLAM

Many SLAM techniques suffer from the computational complexity of maintaining a growing state. Either more particles are required to track more landmarks, or the state of the system increases with each landmark. At best, the increased state causes quadratic growth in computational complexity. To address in this problem, Forgetful SLAM prunes out landmarks that are no longer visible or have not been visible for some set time once it has reached a set cap for landmarks that can be tracked. MonoSLAM briefly talks about state reduction to improve speed, but instead limits the method to only a single room where all features are revisited often[46]. By specifying a rule to forget landmarks, the chosen SLAM method can operate in constant time for each iteration and remain computationally tractable.

The shortcoming of this method is that by forgetting landmarks, the algorithm no longer improves them over time. This piece should be thought of as a high-resolution local tracking stage. The shortcoming is addressed in the next chapter where another algorithm is built on top of Forgetful SLAM that promotes some “good” landmarks. Landmarks with small covariance matrices are promoted to higher-level algorithms, which can use these “forgotten” landmarks for higher-level corrections. The combination of the two methods provides a hierarchical structure for developing large maps in a computationally tractable way.

6.1. Method Details

Until a landmark is revisited, standard SLAM routines will improve the locations of previously seen landmarks in a diminishing way that yields very little benefit. Sparsification is based on a similar concept, where the map is broken into smaller maps since the off diagonals of the state information matrix approaches zero as time goes on[50]. Instead of breaking the map into smaller maps and reassembling them, we simply run an online Forgetful SLAM that discards landmarks that are no longer seen.

Similar to the Robust Kalman Filter, landmarks can be removed from the state under a certain filtering rule. Once removed, the state becomes much smaller and becomes more manageable. A simple rule we implemented is to forget a landmark once it is no longer visible. LK-SURF is able to track features in 3D space for as long as the feature is visible; this removes the computational complexity of associating prior landmarks from this local tracking stage. Forgetful SLAM does not perform feature association, it assumes another method like LK-SURF will track feature IDs. By forgetting features that are no longer visible, Forgetful SLAM reduces the state to the smallest size necessary to predict the given observations. Since LK-SURF will not give revisited features the same ID once it was lost, there is no need for Forgetful SLAM to remember the lost feature. Erroneous features removed by the Robust Kalman Filter method are tracked by Forgetful SLAM in a list so they can be prescreened from subsequent iterations of the filter. This ensures that incorrectly tracked features will not accidentally be reintroduced or promoted to higher level SLAM. This was addressed due to issues with LK-SURF that lead to the occasional incorrectly tracked feature.

The standard state space model and special notation used in Forgetful SLAM is shown in Table 19. This notation keeps track of feature IDs without placing them incorrectly into the state. Unlike positions and headings, feature IDs cannot be improved upon and therefore should not be included in the state vector or covariance matrix since there is no certainty or corrections available. Table 19 takes into account the system dynamics, sensor observation functions, and landmark observation functions.

System Equations:	$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$ $\mathbf{z}_k^i = h(\mathbf{x}_k) + \mathbf{v}_k$ $\mathbf{z}_k^i = h_{feature}(\mathbf{x}_k, \mathbf{m}^i) + \mathbf{v}_k^i$ $\mathbf{m}^i = h_{feature}^{-1}(\mathbf{x}_k, \mathbf{z}_k^i)$ $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$ $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$ $\mathbf{v}_k^i \sim N(\mathbf{r}(\mathbf{x}_k, \mathbf{m}^i), \mathbf{R}(\mathbf{x}_k, \mathbf{m}^i))$
Jacobian Linearization:	$\mathbf{F}_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right _{\mathbf{x}_{k-1}, \mathbf{u}_k} \quad \mathbf{H}_k = \left. \frac{\partial h}{\partial \mathbf{x}} \right _{\mathbf{x}_{k-1}} \quad \mathbf{H}_k^i = \left. \frac{\partial h}{\partial \mathbf{x}} \right _{\mathbf{x}_{k-1}, \mathbf{m}^i}$ $\mathbf{H}_{mx}^j = \left. \frac{\partial h^{-1}}{\partial \mathbf{x}} \right _{\mathbf{x}_{k-1}, \mathbf{z}^j} \quad \mathbf{H}_{mz}^j = \left. \frac{\partial h^{-1}}{\partial \mathbf{z}} \right _{\mathbf{x}_{k-1}, \mathbf{z}^j}$
State Notation with Maps:	<p style="text-align: center;">State: \mathbf{X} Covariance: \mathbf{P} Pose: \mathbf{x}_k Landmark: \mathbf{m}^i</p> $\mathbf{X}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{m}^1 \\ \vdots \\ \mathbf{m}^n \end{bmatrix}$
List Notation:	$\langle \rangle$

Table 19 - System equations and notation

The Forgetful SLAM algorithm is composed of several stages. The first and key part of Forgetful SLAM is the removal of features. Features are removed by a predetermined rule, in this case once they are not observed. Table 20 shows the algorithm for removing features from current state and map. This algorithm is modified

in a later section (see Section 7.1: Modifications of Forgetful SLAM for HAR-SLAM). The modification is made in order to preserve information from the feature removal for use in the global map. The feature removal algorithm simply finds all features not observed and creates a reduced identity matrix to trim down both the state mean vector and state covariance matrix.

RemoveLostFeatures

Input: $(\mathbf{X}, \mathbf{P}, \langle ID_{current} \rangle, \langle ID_{observed} \rangle)$

Output: $(\mathbf{X}_{reduced}, \mathbf{P}_{reduced}, \langle ID_{reobserved} \rangle)$:

$$\langle ID_{reobserved} \rangle = \langle ID_{current} \rangle \cap \langle ID_{observed} \rangle$$

$$\langle ID_{lost} \rangle = \bigcup_{i \in \langle ID_{current} \rangle} i \notin \langle ID_{observed} \rangle$$

$\mathbf{F} = identity$

For each ID in $\langle ID_{lost} \rangle$ find the index j in $\langle ID_{current} \rangle$

$$\mathbf{F}_j = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ddots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \vdots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Elements before index j
State pertaining to index j
Elements after index j

$\mathbf{F} = \mathbf{F}_j \mathbf{F}$

$\mathbf{X}_{reduced} = \mathbf{F} \mathbf{X}$

$\mathbf{P}_{reduced} = \mathbf{F} \mathbf{P} \mathbf{F}^T$

Table 20 - Algorithm for removing lost features

The next part of Forgetful SLAM is common to many SLAM algorithms. This is the addition of newly seen landmarks or features into the current map as shown in Table

21. This stage occurs before any Kalman updates in order to allow the landmarks to develop a cross-covariance from the common observation. An inverse observation function is expected to exist, where given a pose and measurement, a landmark can be created. A complication occurs with nonzero mean error distributions, which is the case for our optical ranging system (see Chapter 10: Calibration and Modeling). Technically, a limit needs to be calculated in order to predict where a landmark is located given a nonzero mean estimate. Once the landmark has an estimated location it can be initialized with the measurement noise. To approximate this, a two-step iteration has been implemented, and in practice has provided sufficient accuracy. $\hat{\mathbf{z}}$ is the result of the nonzero mean calculated from a one stage guess of the landmark location assuming the measured range is perfect. The state and covariance are expanded using $\hat{\mathbf{z}}$ as the parameter passed to the inverse observation function.

<p>AddNewFeatures</p> <p>Input: $(\mathbf{X}, \mathbf{P}, \langle ID_{reobserved} \rangle, \langle \mathbf{Z} \rangle, \langle ID_{observed} \rangle, h_{feature}^{-1}, \mathbf{r}, \mathbf{R})$</p> <p>Output: $(\mathbf{X}_{expanded}, \mathbf{P}_{expanded}, \langle ID_{expanded} \rangle)$:</p> $\langle ID_{new} \rangle = \bigcup_{i \in \langle ID_{observed} \rangle} i \notin \langle ID_{reobserved} \rangle$ $\langle ID_{expanded} \rangle = \langle ID_{reobserved} \rangle \cup \langle ID_{new} \rangle$ $\mathbf{X}_{expanded} = \mathbf{X}$ $\mathbf{P}_{expanded} = \mathbf{P}$ <p>For each ID_j in $\langle ID_{new} \rangle$</p> $\hat{\mathbf{z}} = \mathbf{z}^j - \mathbf{r}(\mathbf{x}, h_{feature}^{-1}(\mathbf{x}, \mathbf{z}^j))$ $\mathbf{X}_{expanded} = \begin{bmatrix} \mathbf{X}_{expanded} \\ h_{feature}^{-1}(\mathbf{x}, \hat{\mathbf{z}}) \end{bmatrix}$ $\mathbf{P}_{expanded} = \begin{bmatrix} \mathbf{P}_{expanded} & 0 \\ 0 & \mathbf{H}_{mz}^j \mathbf{R}(\mathbf{x}, h_{feature}^{-1}(\mathbf{x}, \hat{\mathbf{z}})) \mathbf{H}_{mz}^{jT} + \mathbf{H}_{mx}^j \mathbf{P}_{pose} \mathbf{H}_{mx}^{jT} \end{bmatrix}$

Table 21 - Algorithm for adding new features

The entire Forgetful SLAM algorithm is shown in Table 22. It contains two feature management stages. In the first stage, all observed features are prescreened against a known list of landmarks that are not reliable. Next unseen features are removed using the algorithm from Table 20. The next stage is the standard prediction stage using the same steps as the Extended Kalman Filter. This stage takes into account the dynamics of the system. The normal Kalman update stage comes next for all non-landmark observations. Next, the robust observation stage comes from the Robust Kalman Filter in an earlier chapter. In this step, all landmarks are weighted based on their difference of expected value and outliers are removed.

<p>ForgetfulSLAM</p> <p>Input: $(\mathbf{X}_{k-1}, \mathbf{P}_{k-1}, \langle ID_{state} \rangle_{k-1}, \mathbf{z}_k, \langle \mathbf{Z} \rangle_k, \langle ID_{observed} \rangle_k, \mathbf{u}_k)$</p> <p>Output: $(\mathbf{X}_k, \mathbf{P}_k, \langle ID_{state} \rangle_k)$:</p> <p>Feature Management Stage 1: Remove observations previously marked in $\langle ID_{remove} \rangle$ $(\mathbf{X}_{reduced}, \mathbf{P}_{reduced}, \langle ID_{reobserved} \rangle)$ $= RemoveLostFeatures(\mathbf{X}_{k-1}, \mathbf{P}_{k-1}, \langle ID_{state} \rangle_{k-1}, \langle ID_{observed} \rangle_k)$</p> <p>Prediction Stage: $\mathbf{X}_{predict} = \begin{bmatrix} f(\mathbf{x}_{k-1}, \mathbf{u}_k) \\ \mathbf{m}^1 \\ \vdots \\ \mathbf{m}^n \end{bmatrix}$ $\mathbf{P}_{predict} = \begin{bmatrix} \mathbf{F}_k & 0 & \dots \\ 0 & 1 & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} \mathbf{P}_{reduced} \begin{bmatrix} \mathbf{F}_k^T & 0 & \dots \\ 0 & 1 & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_k & 0 & \dots \\ 0 & 0 & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix}$</p> <p>Update Stage 1: Update $\mathbf{X}_{predict}$ and $\mathbf{P}_{predict}$ using a standard Extended Kalman Update with non-landmark based sensor observations and dynamics</p>
--

Robust Observation Stage:

$$\forall i \in \text{observed features} \begin{cases} \tilde{\mathbf{z}}_k^i = h_{feature}(\hat{\mathbf{x}}_k, \mathbf{m}^i) + \mathbf{r}(\hat{\mathbf{x}}_k, \mathbf{m}^i) \\ \mathbf{K} = \mathbf{P}\mathbf{H}_i^T (\mathbf{H}_i\mathbf{P}\mathbf{H}_i^T + \mathbf{R}_k)^{-1} \\ \mathbf{x}_i = \mathbf{K}(\mathbf{z}_k^i - \tilde{\mathbf{z}}_k^i) \end{cases}$$

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad \tilde{\mathbf{X}} = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$$

\mathbf{v} = largest eigenvector by eigenvalue of $(\tilde{\mathbf{X}})$

$$\forall i \in \text{observed features} \{ w_i = \mathbf{v}^T \mathbf{x}_i \}$$

Get $\langle ID_{reduced} \rangle$, and $\langle ID_{remove} \rangle$ from the Robust Kalman Filter Removal Phase

$$(\mathbf{X}_{predict}, \mathbf{P}_{predict}, \langle ID_{state} \rangle_k)$$

$$= \text{RemoveLostFeatures}(\mathbf{X}_{predict}, \mathbf{P}_{predict}, \langle ID_{reobserved} \rangle, \langle ID_{reduced} \rangle)$$

Update Stage 2:

For each feature in $\langle ID_{reduced} \rangle$:

$$\tilde{\mathbf{z}}_k^i = h_{feature}(\hat{\mathbf{x}}_k, \mathbf{m}^i) + \mathbf{r}(\hat{\mathbf{x}}_k, \mathbf{m}^i)$$

$$\mathbf{F}_j = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \ddots & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & \ddots & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \vdots & \ddots & \ddots & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 & \dots & 0 \end{bmatrix}$$

Non-feature elements
Features before j
State pertaining to index j
Features after j

$$\mathbf{K} = \mathbf{P}_{predict} \mathbf{F}_j^T \mathbf{H}_k^j (\mathbf{H}_k^j \mathbf{F}_j \mathbf{P}_{predict} \mathbf{F}_j^T \mathbf{H}_k^j + \mathbf{R}(\hat{\mathbf{x}}_k, \mathbf{m}^j))^{-1}$$

$$\mathbf{X}_{predict} = \mathbf{X}_{predict} + \mathbf{K}(\mathbf{z}_k^i - \tilde{\mathbf{z}}_k^i)$$

$$\mathbf{P}_{predict} = \mathbf{P}_{predict} - \mathbf{K}\mathbf{H}_k^j \mathbf{F}_j \mathbf{P}_{predict}$$

Feature Management Stage 2:

$$(\mathbf{X}_k, \mathbf{P}_k, \langle ID_{state} \rangle_k)$$

$$= \text{AddNewFeatures}(\mathbf{X}_{predict}, \mathbf{P}_{predict}, \langle ID_{state} \rangle_{predict}, \langle \mathbf{Z} \rangle_k, \langle ID_{observed} \rangle_k, h_{feature}^{-1}, \mathbf{r}, \mathbf{R})$$

Table 22 - Algorithm of Forgetful SLAM

The second to last stage in Forgetful SLAM is the update stage. This stage is the same as the EKF SLAM where robust observations are applied to correct the landmark locations. Finally, the second feature management stage updates the results with newly observed landmarks as the resulting state vector and state covariance is returned and an entire iteration if Forgetful SLAM is complete.

6.2. Advantages of Forgetful SLAM

The primary advantage of Forgetful SLAM over traditional SLAM techniques is the computational complexity of the algorithm remains constant. It is entirely dependent on the observed features and its complexity does not grow over time with newly observed features. Since this algorithm can run in constant time, it allows other, more advanced algorithms to use the results of this method.

This algorithm can be thought of as a sensor fusion technique, where system states, landmarks, and sensors are all brought together to make the best possible estimate of the state and the map. In an online manner, it correlates all measurements and landmarks with each other, giving a much better result than the sensor just by itself.

Given that the Robust Kalman Filter performs better than the EKF (see Section 5.4: Analysis) when used in Forgetful SLAM, and without loop-closing, the standard EKF SLAM will have no advantage over Forgetful SLAM. The main advantage of Forgetful SLAM is the reduced computational complexity. Figure 47 shows the computation time it takes to perform Forgetful SLAM versus EKF SLAM. Forgetful SLAM remains near constant per iteration depending on the number of visible landmarks.

This becomes a linear rise for total time per iteration. EKF SLAM grows quadratically over time [27], which results in a cubic rise for total time per iteration. The example in Figure 47 shows the computation time for the path drawn in Figure 46. The Forgetful SLAM routine shows linear rise in total time per iteration, and EKF SLAM shows cubic growth. Due to time constraints, only a single comparison was made.

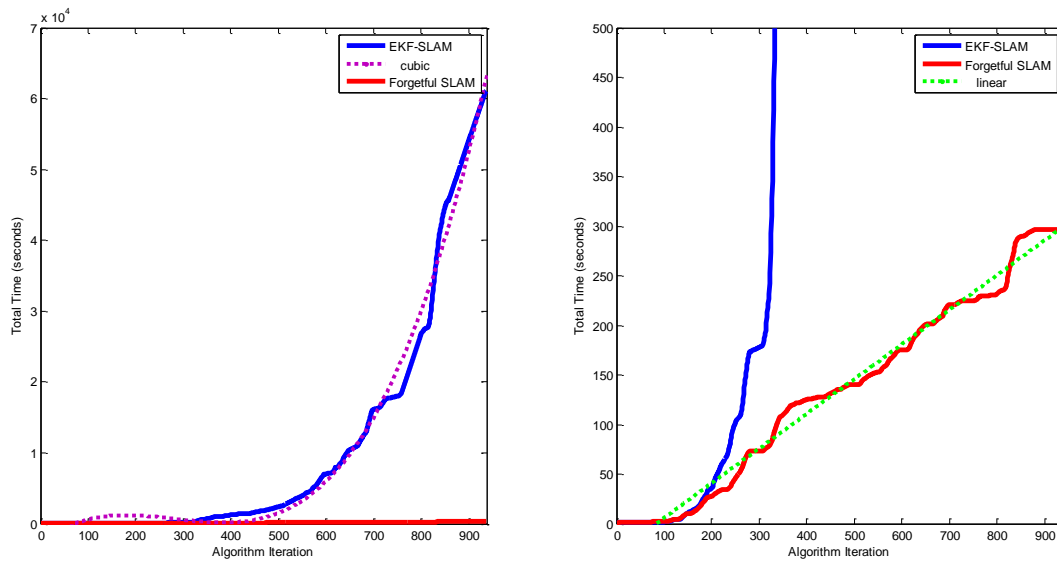


Figure 47 - Comparison of EKF SLAM computation cost vs. Forgetful SLAM computation cost

The example clearly shows the difference in computation time. EKF-SLAM took 16.98 hours to complete the sample map. Forgetful SLAM took 4.95 minutes to complete the same map. Both algorithms used the Robust Kalman Filter to eliminate outliers, thus both algorithms had the same number of landmarks. Furthermore, EKF-SLAM with outlier removal produced worse results on this example than Forgetful SLAM, even with outlier rejection. Figure 48 shows a comparison of path generated by Forgetful SLAM, EKF-SLAM, and the estimate of the ground truth. Forgetful SLAM generated 1.35% error over distance travelled. EKF-SLAM with outlier removal generated 9.05% error over distance travelled.

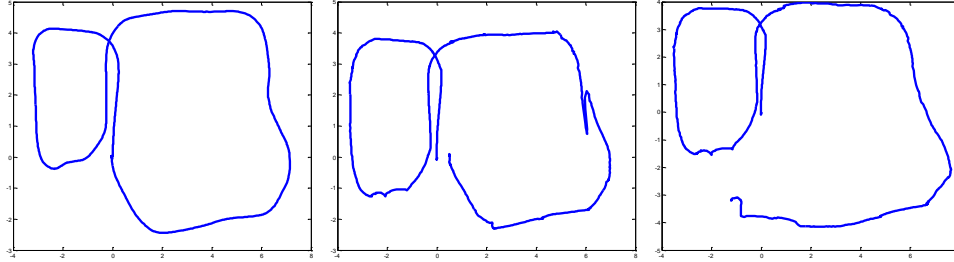


Figure 48 - Comparison of Forgetful SLAM path to EKF-SLAM path.
 Left: estimate of true path; Center: Forgetful SLAM path; Right: EKF-SLAM path

6.3. Disadvantage of Forgetful SLAM

Forgetful SLAM has a few disadvantages when compared to other SLAM techniques. The biggest disadvantage is that it has no capability to close-the-loop, meaning that revisiting landmarks does not improve the estimate of the state as it would with other SLAM techniques. Forgetting the map does not only ruin accuracy when revisiting landmarks, but doesn't improve the entire map as time goes on; only observed features are improved, forgotten features are left alone.

It is due to these disadvantages that another SLAM technique needs to run on top of Forgetful SLAM. However, in its current stage, Forgetful SLAM does not lend itself well to allowing the entire map to be recovered or modified. In a later section (see Section 7.1: Modifications of Forgetful SLAM for HAR-SLAM), modifications to Forgetful SLAM are developed that allow it to become useful to other SLAM techniques.

6.4. Modification for Neural Net Based Models

In order to handle system models from neural net based modeling techniques, the Unscented Transform is used. Neural net based models make it difficult to calculate

Jacobians, which makes the Unscented Transform desirable, as it does not require any Jacobians, but instead linearizes the system noise, not the system model. Table 23 shows the replacement for the prediction stage in Table 22. The new method is based on the Unscented Transform, while the original technique uses the Extended Kalman Filter technique for state prediction.

<p>UnscentedStatePrediction</p> <p>Input: $(\mathbf{X}_{k-1}, \mathbf{P}_{k-1}, \mathbf{U}_k, f)$</p> <p>Output: $(\mathbf{X}_k, \mathbf{P}_k)$</p> <p>Optimal Constants for Gaussians: $\alpha = 0.5, \beta = 2, \kappa = 0, \lambda = \alpha^2(N + \kappa) - N, N = \text{size of state}$</p> <p>Calculate Sigma Points:</p> $\mathbf{R} = \text{chol}(\mathbf{P}_{k-1})\sqrt{N + \lambda}$ $\Sigma_0 = \mathbf{X}_{k-1}$ $\Sigma_{i=1..N} = \mathbf{X}_{k-1} + \mathbf{R} \begin{bmatrix} 0 & 1 & 0 \\ \langle i & i^{\text{th}} \text{ column} & \rangle i \end{bmatrix}$ $\Sigma_{i=N+1..2N} = \mathbf{X}_{k-1} - \mathbf{R} \begin{bmatrix} 0 & 1 & 0 \\ \langle i/2 & (i/2) \text{ column} & \rangle (i/2) \end{bmatrix}$ <p>Propagate Sigma Points:</p> $\Sigma_{i=0..2N} = \begin{bmatrix} f(\mathbf{x}_{\Sigma_i}, \mathbf{u}_k) \\ \mathbf{m}_{\Sigma_i}^1 \\ \vdots \\ \mathbf{m}_{\Sigma_i}^n \end{bmatrix}$ <p>Calculate Sigma Properties:</p> $\mathbf{X}_k = \frac{\lambda}{\lambda + N} \Sigma_0 + \sum_{i=1}^{2N} \left(\frac{\Sigma_i}{2(\lambda + N)} \right)$ $\mathbf{P}_k = \left(1 - \alpha^2 + \beta + \frac{\lambda}{\lambda + N} \right) (\Sigma_0 - \mathbf{X}_k)(\Sigma_0 - \mathbf{X}_k)^T + \sum_{i=1}^{2N} \left(\frac{(\Sigma_i - \mathbf{X}_k)(\Sigma_i - \mathbf{X}_k)^T}{2(\lambda + N)} \right)$

Table 23 - State prediction using Unscented Transform

Instead of using the plain Unscented Transform, we could also use the prediction stage from CUMRF, as described in an earlier chapter. This method uses the Unscented Transform as well, but also handles non-Gaussian noise through a Gaussian Mixture Model. This allows us to have a more accurate prediction stage, and with the consolidation part of CUMRF, we are left with only a single Gaussian, making the rest of the computations more efficient.

6.5. Applications

Forgetful SLAM was created in order to fuse sensors and landmarks in constant computational effort for local tracking. Current SLAM techniques cannot account for the unexpected errors in observations as elegantly as the Robust Kalman Filter stage of Forgetful SLAM. The technique was developed with HAR-SLAM in mind, where the features and poses could be organized in chains instead of large matrices. This technique improves the handling of inexpensive and inaccurate sensor measurements to produce correlated and accurate results. The next chapter includes modifications to Forgetful SLAM in order to retrieve correlations to past poses and landmarks. These modifications allow Forgetful SLAM to be used by other high-level SLAM algorithms.

CHAPTER 7: HIERARCHICAL ACTIVE RIPPLE SLAM

SLAM routines are either active or inactive. Kalman based SLAM routines are considered active since each landmark and robot state is updated at every sample. GraphSLAM is considered an inactive or lazy SLAM, since it can be solved in a batch process off-line. Hierarchical Active Ripple SLAM (HAR-SLAM) uses two layers for SLAM. The first layer is a modified Forgetful SLAM routine, where sensor measurements, robot poses, and landmarks are associated, improved, and then forgotten when they are out of visual range. In the second layer, the forgotten poses and landmarks are extracted in a similar method to GraphSLAM using properties of the Information matrix. Each pose has a correlation to the next pose, and as landmarks are removed from Forgetful SLAM, their correlations are tied to their last robot pose association only. The past poses and landmarks form a connected graph, where all landmarks are connected to a pose, and all poses form a chain.

As new poses are produced from Forgetful SLAM, the entire system is updated in a ripple technique. The ripple technique involves updating each pose in the connected chain in series, and updating the connecting landmarks. Each pose is updated by a linked neighbor (pose or landmark) with a Kalman-like Filter using the pose state, the pose covariance matrix, the linked neighbor's state, the linked neighbor's covariance matrix, and the cross-covariance matrix between the pose and neighbor. Each landmark is updated by an attached pose through a Kalman-like Filter using the landmark state and

covariance, the pose state and covariance, and the cross correlation matrix between the pose and landmark. There is a state vector and covariance matrix per pose, a state vector and covariance matrix per landmark, and a cross-covariance matrix per link. This structure makes storage, and computation cost linear with number of landmarks and poses.

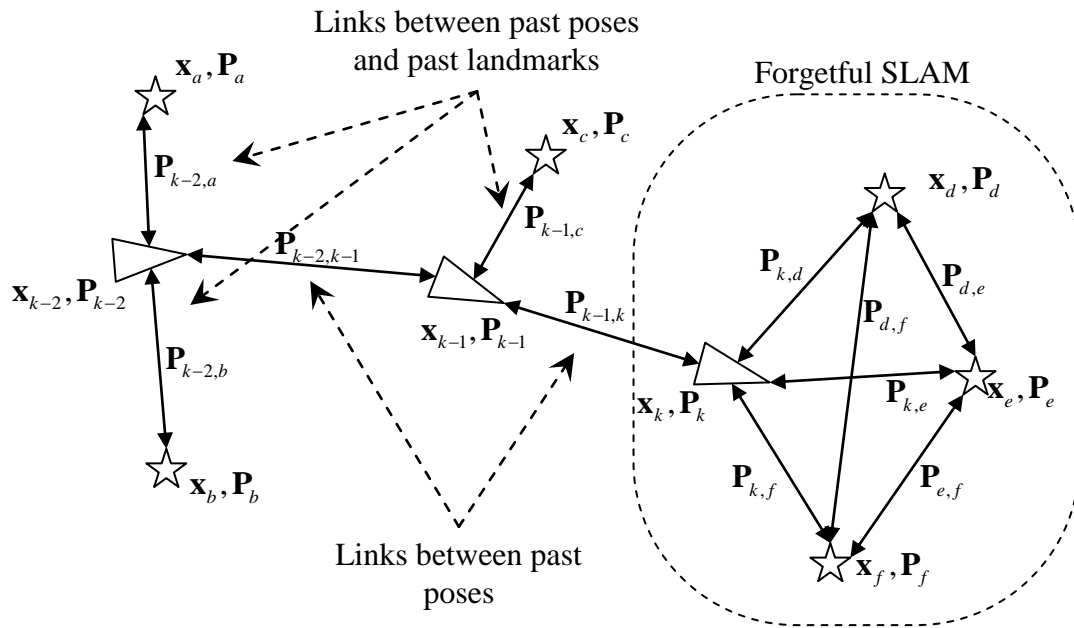


Figure 49 - Overview of HAR-SLAM. Forgetful SLAM operating on current information. Links are established between past poses and landmarks.

Figure 49 shows the conceptual structure of HAR-SLAM. The section labeled Forgetful SLAM shows the most recent data being affected. Forgetful SLAM only correlates and updates visible landmarks. Each forgotten landmark is attached to a past pose. Each pose is connected in series over time. Updates in HAR-SLAM travel along the connections shown in Figure 49.

HAR-SLAM is not guaranteed to be optimal like the Kalman Filter. HAR-SLAM operates on systems with nonlinear dynamics, like EKF-SLAM, FastSLAM, and

GraphSLAM, making an optimal solution infeasible in most cases. HAR-SLAM uses Forgetful SLAM, which employs the Robust Kalman Filter, making the system resistant to outliers. Remembering the entire path instead of only the best estimate allows HAR-SLAM to recover from errors, unlike EKF-SLAM. This property of remembering all poses and linking landmarks only to a single pose allows multiple robots to link maps together and asynchronously update portions of the map.

The updating mechanism of HAR-SLAM is similar to a physics engine in modern computer games, where every element in the engine is linked to its neighbors and updates upon some defined physics rule. In HAR-SLAM, each pose and landmark is updated depending only on its direct links using Kalman gains. Eventually, given enough steps, every component will be updated. Like GraphSLAM, HAR-SLAM will eventually converge to a solution. Unlike GraphSLAM, HAR-SLAM actively updates each component, and convergence is immediate for each updated pose and landmark in the map. HAR-SLAM, like physics engines, can be implemented in parallel, giving an incredible speed boost with today's processors, but that is beyond the scope of this dissertation. Showing HAR-SLAM converges when updated in parallel is also beyond the scope of this dissertation.

7.1. Modifications of Forgetful SLAM for HAR-SLAM

In order for HAR-SLAM, or any other high level SLAM technique, to use Forgetful SLAM, the forgotten features need to be recoverable. A correlation matrix is generated per lost feature that relates the feature to the previous pose. Ultimately, a

series of Kalman Filters can update these lost features from their past poses. In order to modify Forgetful SLAM properly, only the first function call to feature removal is replaced. It is not advantageous to recall features removed due to robustness issues. Table 25 shows the new removal algorithm, which returns a list of removed IDs, each features mean, each features covariance matrix, the past pose mean, and past pose covariance matrix. Table 24 shows the notation used in the new removal algorithm.

Matrix Notation:

$$\mathbf{G}_j = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 1 & \ddots & \vdots & 0 & \dots & 0 \\ \vdots & \dots & \vdots & \vdots & \ddots & \ddots & 0 & \vdots & \dots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} \quad \mathbf{G}_{pose} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \vdots & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots & \dots & \vdots \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

Elements before index j
State pertaining to index j
Elements after index j
State pertaining pose
State after pose

$$\mathbf{F}_j = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \ddots & \vdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \quad \hat{\mathbf{G}}_j = \begin{bmatrix} \mathbf{G}_{pose} \\ \mathbf{G}_j \end{bmatrix}$$

Elements before index j
State pertaining to index j
Elements after index j

$$\hat{\mathbf{F}}_j = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 & \ddots & \vdots & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & & \vdots & \vdots & \ddots & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ddots & \vdots \\ \vdots & & \vdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

State pertaining to pose
Elements before index j
State pertaining to index j
Elements after index j

Table 24 - Matrix notation for feature removal

The algorithm presented in Table 25 shows the detail of recovering the information matrix associated with the current pose and lost landmarks. Since this is done before any prediction or updates, the pose and map are actually from the previous iteration. The algorithm correlates the past landmark with the past pose. The covariance matrix is generated in the same way GraphSLAM removes landmark links [51], except instead of removing a single landmark to generate new links, we remove all but the target landmark and leave just the pose and single landmark. Unlike GraphSLAM, covariance matrices are generated instead of information matrices; the canonical form GraphSLAM uses is computationally impractical for active updates. The landmark and pose covariance matrices are saved for later updates.

RemoveLostFeatures	
Input: $(\mathbf{X}, \mathbf{P}, \langle ID_{current} \rangle, \langle ID_{observed} \rangle)$	
Output: $(\mathbf{X}_{reduced}, \mathbf{P}_{reduced}, \langle ID_{reobserved} \rangle, \langle \mathbf{X}_{lost} \rangle, \langle \mathbf{P}_{lost} \rangle, \langle \mathbf{P}_{cross} \rangle, \langle ID_{lost} \rangle, \mathbf{X}_{pose}, \mathbf{P}_{pose})$	
$\langle ID_{reobserved} \rangle = \langle ID_{current} \rangle \cap \langle ID_{observed} \rangle$	
$\langle ID_{lost} \rangle = \bigcup_{i \in \langle ID_{current} \rangle} i \notin \langle ID_{observed} \rangle$	
$\mathbf{F} = \text{identity}$	
For each ID in $\langle ID_{lost} \rangle$ find the index j in $\langle ID_{current} \rangle$	
$\mathbf{F} = \mathbf{F}_j \mathbf{F}$	
$\mathbf{P}_f = \mathbf{G}_j \mathbf{P} \mathbf{G}_j^T$	Add \mathbf{P}_f to $\langle \mathbf{P}_{lost} \rangle$
$\mathbf{P}_c = \mathbf{G}_{pose} \mathbf{P} \mathbf{G}_j^T$	Add \mathbf{P}_c to $\langle \mathbf{P}_{cross} \rangle$
$\mathbf{X}_f = \mathbf{G}_j \mathbf{X}$	Add \mathbf{X}_f to $\langle \mathbf{X}_{lost} \rangle$
$\mathbf{X}_{pose} = \mathbf{G}_{pose} \mathbf{X}$	
$\mathbf{P}_{pose} = \mathbf{G}_{pose} \mathbf{P} \mathbf{G}_{pose}^T$	
$\mathbf{X}_{reduced} = \mathbf{F} \mathbf{X}$	
$\mathbf{P}_{reduced} = \mathbf{F} \mathbf{P} \mathbf{F}^T$	

Table 25 - Algorithm for retaining lost features

Adapting techniques from GraphSLAM again, a covariance matrix is created between the previous pose and the current pose. This covariance matrix is created only using the re-observed features. This correlation algorithm can run at various spots in Forgetful SLAM. However, to get the entire benefit of the Robust Kalman Filter and to get the benefit of the Kalman Filter's ability to generate cross correlations during the update phase, the correlation algorithm runs after the second update stage. The operation is placed before new landmarks are added.

Augmented State Function:	
$\tilde{f}(\mathbf{x}, \mathbf{u}) =$	$\begin{bmatrix} f(\mathbf{x}, \mathbf{u}) \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{bmatrix}$
$\tilde{\mathbf{Q}} =$	$\begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 0 \end{bmatrix}$

Table 26- Augmented state functions for pose correlation

The algorithm combines the re-observed features into the cross-covariance between the past and present pose. This allows modifications of the current pose to propagate to past poses and eventually to past landmarks or features. However, if we want to correlate the past pose and current pose after the Kalman update, we need to augment the state to include the past pose. A few simple changes to the system equations can handle the augmented state. Table 26 shows the augmented state function that takes into account the past pose into the state allowing correlation to be done after the Kalman update. This is a simple expansion of the state, where the current state is set as the previous state. In practice, this expansion for $\tilde{\mathbf{Q}}$ can yield unstable results. If this is the

case, we simply inflate $\tilde{\mathbf{Q}}$ a tiny amount in the bottom right block of zeros to prevent a determinant of zero.

Once the system is augmented, the correlation algorithm shown in Table 27 runs after the last update stage. The commonly seen features correlate the past and current pose in Forgetful SLAM. Given active updates, the reduced information matrix correlating the current and past pose is unnecessary as in GraphSLAM [51] or in the Sparse Extended Information Filter [27].

CorrelatePoses	
Input: $(\tilde{\mathbf{P}}_k)$	Output: $(\mathbf{P}_k, \mathbf{P}_{k-1}, \mathbf{P}_{cross})$
$\tilde{\mathbf{P}}_k = \begin{bmatrix} \tilde{\mathbf{P}}_k & & \tilde{\mathbf{P}}_{k,features} \\ \tilde{\mathbf{P}}_{k,k-1}^T & \tilde{\mathbf{P}}_{k-1} & \tilde{\mathbf{P}}_{k-1,features} \\ \tilde{\mathbf{P}}_{k,features}^T & \tilde{\mathbf{P}}_{k-1,features}^T & \tilde{\mathbf{P}}_{features} \end{bmatrix}$	
$\mathbf{P}_k = \tilde{\mathbf{P}}_k$	
$\mathbf{P}_{k-1} = \tilde{\mathbf{P}}_{k-1}$	
$\mathbf{P}_{cross} = \tilde{\mathbf{P}}_{k,k-1}$	

Table 27 - Algorithm for creating pose cross variance from augmented state

After the poses are correlated, we reduce the state size again to remove the past state. We need to extract the newly placed past pose as well. This simple extraction is shown in Table 28. Inevitably, there is some uncertainty associated with the past pose, and the observations that updated the current pose will update the past pose. This newly updated past pose will update the existing chain of poses. This update phase will be described in the next section.

ReduceStateSize									
Input: $(\tilde{\mathbf{X}}_k, \tilde{\mathbf{P}}_k)$					Output: $(\mathbf{X}_k, \mathbf{X}_{k-1}, \mathbf{P}_k)$				
$\mathbf{F}_k =$	$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & \ddots & \vdots & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots & & \vdots & 0 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \ddots \\ 0 & 0 & 0 & 0 & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$ <div style="display: flex; justify-content: space-around; font-size: small; margin-top: 5px;"> State pertaining to pose at time k State pertaining to pose at k-1 Remainder of the map </div>								
$\mathbf{F}_{k-1} =$	$\begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 & \ddots & \vdots & 0 & 0 & 0 \\ \vdots & & \vdots & \vdots & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ddots \\ \vdots & & \vdots & 0 & 0 & 0 & 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$ <div style="display: flex; justify-content: space-around; font-size: small; margin-top: 5px;"> State pertaining to pose at k State pertaining to pose at time k-1 Remainder of the map </div>								
$\mathbf{X}_k = \mathbf{F}_k \tilde{\mathbf{X}}_k$									
$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}_k \mathbf{F}_k^T$									
$\mathbf{X}_{k-1} = \mathbf{F}_{k-1} \tilde{\mathbf{X}}_k$									

Table 28 - Reduction of the augmented state

7.2. Method Details

HAR-SLAM has two main stages, the first is Forgetful SLAM, and the second is a rippling update. Forgetful SLAM has to be modified to include a link between forgotten landmarks and past poses. This is done in Table 25, where removed landmarks are correlated only to the past pose. These correlations are important for updating

landmarks once the pose changes, and for revising landmarks to understand how to modify the past pose. Past poses have correlations to each other forming a chain; when a pose is modified, an update routine modifies all connecting neighbors. The rippling update is simply the propagation of updates to every connected pose and landmark in a directed graph. The correlation algorithm is shown in Table 27, where an augmented state is updated and the information matrix between the past and current pose is extracted. It is important to augment the system so the newly observed landmarks can be properly associated with the current pose and at the same time update the past pose correlation.

7.2.1. Standard Update

The second stage to HAR-SLAM is the actual update of the chain of poses and landmarks. After a pose and attached landmarks are extracted from the first stage of Forgetful SLAM, the pose is immediately updated by the end of an iteration of Forgetful SLAM. This causes the first and most common update in the system, which is an update in pose. The algorithm outlined in Table 29 shows how we update a destination pose from a source pose. For example, after \mathbf{X}_k is created, \mathbf{X}_{k-1} is updated. The update is then rippled as a change from \mathbf{X}_{k-1} to \mathbf{X}_{k-2} , and from \mathbf{X}_{k-1} to all attached landmarks.

UpdateLinkState
Input: $(\mathbf{X}_{source}, \mathbf{X}_{destination}, \mathbf{X}_{source}^{new}, \mathbf{P}_{source}, \mathbf{P}_{destination}, \mathbf{P}_{cross})$
Output: $(\mathbf{X}_{destination}^{new})$
$\mathbf{P} := \begin{bmatrix} \mathbf{P}_{source} & \mathbf{P}_{cross} \\ \mathbf{P}_{cross}^T & \mathbf{P}_{destination} \end{bmatrix}$
$\mathbf{K} = \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1}$
$\mathbf{X}_{destination}^{new} = \mathbf{X}_{destination} + \mathbf{K} (\mathbf{X}_{source}^{new} - \mathbf{X}_{source})$

Table 29 - Algorithm to update past poses and landmarks

The update has a second stage, which updates the covariance matrix of each pose, landmark, and cross-covariance. This is shown in Table 30, where the same Kalman gain is used, but instead updates the covariance matrices. These two algorithms could be combined into a single algorithm; however, later developments require the separation of the two.

UpdateLinkCovariance
Input: $(\mathbf{P}_{source}, \mathbf{P}_{destination}, \mathbf{P}_{cross}, \mathbf{P}_{source}^{new})$
Output: $(\mathbf{P}_{destination}^{new}, \mathbf{P}_{cross}^{new})$
$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{source} & \mathbf{P}_{cross} \\ \mathbf{P}_{cross}^T & \mathbf{P}_{destination} \end{bmatrix}$
$\mathbf{K} = \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1}$
$\mathbf{P}_{destination}^{new} = \mathbf{P}_{destination} + \mathbf{K} (\mathbf{P}_{source}^{new} - \mathbf{P}_{source}) \mathbf{K}^T$
$\mathbf{P}_{cross}^{new} = \mathbf{P}_{source}^{new} \mathbf{K}^T$

Table 30 - Algorithm to update past covariance matrices

Both update algorithms have a relatively small inverse that never grows in dimension. Typically, the source will be the latest state from the Forgetful SLAM routine, and every state that came before, and every landmark will be updated in a sequence. This update is linear with the number of states and landmarks. The direction

is important as the source and destination are updated in different ways. The source state is assumed a known set value, with a known set covariance. If we switch directions between the source and destination, then we merely need to transpose the cross-covariance matrix that relates them together. It is important to note that if the cross-covariance is zero, then the destination will not be updated at all, as there is no relation between the two.

7.2.2. Revisiting Landmark Update

The other important update is when a landmark is observed again. A landmark matching routine can indicate a match at anytime. There is one detrimental fact about observing the same feature again through Forgetful SLAM, and not matching until after it is forgotten. The issue is that the two observations are of the same feature and are correlated, unlike sensor observations, which have no correlation to the state. This presents an interesting update routine, which again uses Kalman-like updates. Table 31 shows the algorithm to update the system given revisited landmarks. There are two odd parts to this algorithm, first the residual covariance noted as \mathbf{S} contains cross-covariance matrices computed by multiplying the chain of cross-covariance matrices and state covariance inverses that connect the two observations of the same landmark. The second oddity is that the Update Link routine is called starting at each feature, but only updates the state vector. The covariance matrices are updated after both features are completely updated, and again ripple from each feature. This is why the update algorithm is presented as two separate versions.

UpdateRevisitedLandmarkInput: $(\mathbf{X}_A, \mathbf{X}_B, \mathbf{P}_A, \mathbf{P}_B, \mathbf{P}_{A \rightarrow B})$ Output: $(\hat{\mathbf{X}}_A, \hat{\mathbf{X}}_B, \hat{\mathbf{P}}_A, \hat{\mathbf{P}}_B)$

$$\mathbf{P}_{A \rightarrow B} := \mathbf{P}_{A,1} (\mathbf{P}_1)^{-1} \mathbf{P}_{1,2} (\mathbf{P}_2)^{-1} \mathbf{P}_{2,3} \dots \mathbf{P}_{n-1,n} (\mathbf{P}_n)^{-1} \mathbf{P}_{n,B}$$

$$\mathbf{S} = \mathbf{P}_A - \mathbf{P}_{A \rightarrow B} - \mathbf{P}_{A \rightarrow B}^T + \mathbf{P}_B$$

$$\hat{\mathbf{X}}_A = \mathbf{X}_A + \mathbf{P}_A \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)$$

$$\hat{\mathbf{X}}_B = \mathbf{X}_B + \mathbf{P}_B \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)$$

Call **UpdateLinkState** on all states connecting feature A to feature B, starting at feature A to state 1, continue until state n and continue to include feature B.

Call **UpdateLinkState** on all states connecting feature B to feature A, starting at feature B to state n, continue until state 1 and continue to include feature A.

$$\hat{\mathbf{P}}_A = \mathbf{P}_A - \mathbf{P}_A \mathbf{S}^{-1} \mathbf{P}_A + \frac{1}{2} (\mathbf{P}_A \mathbf{S}^{-1} \mathbf{P}_{A \rightarrow B}^T + \mathbf{P}_{A \rightarrow B} \mathbf{S}^{-1} \mathbf{P}_A)$$

$$\hat{\mathbf{P}}_B = \mathbf{P}_B - \mathbf{P}_B \mathbf{S}^{-1} \mathbf{P}_B + \frac{1}{2} (\mathbf{P}_B \mathbf{S}^{-1} \mathbf{P}_{A \rightarrow B} + \mathbf{P}_{A \rightarrow B}^T \mathbf{S}^{-1} \mathbf{P}_B)$$

Call **UpdateLinkCovariance** on all states connecting feature A to feature B, starting at feature A to state 1, continue until state n and continue to include feature B.

Call **UpdateLinkCovariance** on all states connecting feature B to feature A, starting at feature B to state n, continue until state 1 and continue to include feature A.

Table 31 - Algorithm to update revisited landmarks

The algorithm presented in Table 31 does not show the final values of the features, poses, or covariance matrices. These values are updated using the Update Link algorithms. A later section will derive and verify these update equations. The algorithm presented assumes features A and B are a successful match, the pose connected to A is arbitrarily labeled 1, the pose connecting to B is arbitrarily labeled n, and pose 1 connects to pose 2, and so on until pose n-1 connects to pose n. Once a feature is matched, the old feature needs to be removed in order to maintain the map being a chain with no loops or complications. This step is not necessary, but makes map storage and updating easier. Table 32 shows the necessary steps to update the map after removing a landmark. The actual removal of the landmark is as simple as just deleting the link, state, and covariance

matrix; no real work is involved. Updating all the cross-covariance matrices is relatively simple given the path between the previously visited and newly revisited landmark is well known. Notice, only cross-covariance matrices are modified; this is because removing a feature has no effect on the pose estimate or landmark estimate, merely affecting the links between landmarks and poses.

<p>Map Reduction (Remove landmark A after matching landmark B to landmark A and updating)</p> <p>Define:</p> $\mathbf{P}_{A \rightarrow B} := \mathbf{P}_{A,1} (\mathbf{P}_1)^{-1} \mathbf{P}_{1,2} (\mathbf{P}_2)^{-1} \mathbf{P}_{2,3} \dots \mathbf{P}_{n-1,n} (\mathbf{P}_n)^{-1} \mathbf{P}_{n,B}$ $\mathbf{P}_{A \rightarrow i} := \mathbf{P}_{A,1} (\mathbf{P}_1)^{-1} \mathbf{P}_{1,2} (\mathbf{P}_2)^{-1} \mathbf{P}_{2,3} \dots \mathbf{P}_{i-1,i}$ $\mathbf{P}_{i \rightarrow B} := \mathbf{P}_{i,i+1} (\mathbf{P}_{i+1})^{-1} \mathbf{P}_{i+1,i+2} (\mathbf{P}_{i+2})^{-1} \mathbf{P}_{i+2,i+3} \dots \mathbf{P}_{n-1,n} (\mathbf{P}_n)^{-1} \mathbf{P}_{n,B}$ $\mathbf{X}_A = \mathbf{X}_B$ $\mathbf{P}_A \approx \mathbf{P}_B$ <p>For all cross-covariance matrices $\mathbf{P}_{i,i+1}$ between landmark A and landmark B excluding $\mathbf{P}_{A,1}$</p> $\mathbf{P}_{i,i+1}^{new} = \mathbf{P}_{i,i+1} + (\mathbf{P}_{A \rightarrow i})^T (\mathbf{P}_A)^{-1} (\mathbf{P}_{i+1 \rightarrow B})^T$
--

Table 32 - Algorithm to update map links after removing a revisited landmark

7.2.3. Multiple Robot Updates

Using multiple robots to generate a map cooperatively requires an additional calculation. The same updates and ripples can be used as previously described in HAR-SLAM; however, there is one extra state. The global coordinate state is required to unite all local robot coordinate systems together. For each robot, there is an entry in the state, and the state is composed of the local rotation to global coordinates and the location of the local origin in the global coordinate system. In the case of a 2D system, we can represent this as a 2D origin and a 1D rotation per robot. In the case of a 3D system, we can represent this as a 3D origin, and 3 orthogonal rotations, or a 4 dimensional

quaternion. A simple translation and rotation function needs to exist to translate the local coordinates into the global coordinate frame.

In order to discover this global coordinate transformation, an Extended Kalman Filter will be used to update the estimate of the global coordinate state. A covariance matrix will be associated with the state; its usefulness will be shown soon. Every time a high-level algorithm decides two or more robots have seen the same feature, then the global coordinate state will be updated. There has to be some initialization of the global coordinate state, minimally let all robots have the same initial global state, and the same large covariance matrix, indicating little knowledge of the true state. Table 33 shows the Kalman Update Algorithm, where the observation of the same feature from two different frames is translated into an error and propagated using the optimal Kalman gain.

UpdateGlobalCoordinates

Input: $(\mathbf{X}_{global}, \mathbf{P}_{global}, \mathbf{X}_{feature}^j, \mathbf{X}_{feature}^k, \mathbf{P}_{feature}^j, \mathbf{P}_{feature}^k)$

Output: $(\mathbf{X}_{global}^{new}, \mathbf{P}_{global}^{new})$

$$\mathbf{X}_{global} := \begin{bmatrix} \mathbf{X}_{origin}^1 \\ \mathbf{X}_{rotation}^1 \\ \vdots \end{bmatrix}$$

$$\mathbf{P}_{global} := \begin{bmatrix} \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \dots \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$R(\dots) :=$ Rotation Matrix to World Coordinates

$$\tilde{\mathbf{y}} = R(\mathbf{X}_{rotation}^k)(\mathbf{X}_{feature}^k - \mathbf{X}_{origin}^k) - R(\mathbf{X}_{rotation}^j)(\mathbf{X}_{feature}^j - \mathbf{X}_{origin}^j)$$

$$\mathbf{R}_i = \frac{\partial}{\partial \mathbf{X}_{rotation}} \left(R(\mathbf{X}_{rotation})(\mathbf{X}_{feature} - \mathbf{X}_{origin}) \right) \Big|_{\substack{\mathbf{X}_{origin} = \mathbf{X}_{origin}^i \\ \mathbf{X}_{rotation} = \mathbf{X}_{rotation}^i \\ \mathbf{X}_{feature} = \mathbf{X}_{feature}^i}}$$

$$\mathbf{F} = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 1 & & \vdots & 0 & \dots & 0 & 0 & 0 & & \vdots & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \ddots & 0 & \vdots & & \vdots & \vdots & & \ddots & 0 & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & & \vdots & 0 & \dots & 0 & 0 & 1 & & \vdots & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \ddots & 0 & \vdots & & \vdots & \vdots & & \ddots & 0 & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

state pertaining to robot j
state pertaining to robot k

$$\mathbf{H}_{global} = \begin{bmatrix} R(\mathbf{X}_{rotation}^j) & -\mathbf{R}_j & -R(\mathbf{X}_{rotation}^k) & \mathbf{R}_k \end{bmatrix}$$

$$\mathbf{H}_i = R(\mathbf{X}_{rotation}^i)$$

$$\mathbf{S} = \mathbf{H}_{global} \mathbf{F} \mathbf{P}_{global} \mathbf{F}^T \mathbf{H}_{global}^T + \mathbf{H}_j \mathbf{P}_{feature}^j \mathbf{H}_j^T + \mathbf{H}_k \mathbf{P}_{feature}^k \mathbf{H}_k^T$$

$$\mathbf{K} = \mathbf{P}_{global} \mathbf{F}^T \mathbf{H}_{global}^T \mathbf{S}^{-1}$$

$$\mathbf{X}_{global}^{new} = \mathbf{X}_{global} + \mathbf{K} \tilde{\mathbf{y}}$$

$$\mathbf{P}_{global}^{new} = \mathbf{P}_{global} - \mathbf{K} \mathbf{H}_{global} \mathbf{F} \mathbf{P}_{global}$$

Table 33 - Kalman Filter for global coordinate update

The algorithm in Table 33 can operate as a standalone algorithm; however, this is not as error resistant as the rest of HAR-SLAM. Feature matching passes through the

low-level Forgetful SLAM routine, which calls the Robust Kalman Filter. This allows the system to remove outliers, which the normal Kalman Filter cannot handle. In order to have the same advantage here, HAR-SLAM for multiple robots needs to keep track of all known features that match between robots. All features commonly observed by robots are passed into the robust update function as described in Table 34. This update function uses a modified Robust Kalman Filter. The version of the Robust Kalman Filter in Table 34 uses the Extended Kalman Filter from Table 33 as a subroutine.

The global coordinate update can operate as frequently or infrequently as desired; however, there is a consequence to its use. After the global coordinate frame is updated, every pair of features that managed to remain in the robust pairing needs to be updated and the resulting effects on each robot's individual map need to be rippled. The initial correction to the features is shown in Table 35, where each pair of features is updated based on the global coordinate state. Corrections are then propagated to the rest of the map using the standard Update Link method described in Table 29 and Table 30. Optionally, when each robot updates its individual map, all affected landmarks could be updated simultaneously using the same method outlined in Table 31, except each landmark would take into consideration the correlation from every other landmark being updated from other robots. This method is computationally more expensive than the simple Update Link method, but is ultimately more accurate. In practice, we can ignore the cross correlation values since they end up being much smaller than the rest of the system, and update often enough that the system settles to the correct answer. This approach is taken in GraphSLAM [51], where iterations and gradient descent methods are used to solve for the map.

RobustGlobalCoordinateUpdate

Input: $(\mathbf{X}_{global}, \mathbf{P}_{global}, \langle \mathbf{X}_{feature}^j, \mathbf{X}_{feature}^k \rangle, \langle \mathbf{P}_{feature}^j, \mathbf{P}_{feature}^k \rangle)$

Output: $(\mathbf{X}_{global}^{new}, \mathbf{P}_{global}^{new})$

Calculate observation weights:

$\forall i \in$ observed feature pairs:

$$\mathbf{x}_i = \text{UpdateGlobalCoordinates} \left(\begin{array}{l} \mathbf{X}_{global}, \mathbf{P}_{global}, \langle \mathbf{X}_{feature}^j, \mathbf{X}_{feature}^k \rangle_i, \\ \langle \mathbf{P}_{feature}^j, \mathbf{P}_{feature}^k \rangle_i \end{array} \right)$$

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad \tilde{\mathbf{X}} = \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$$

\mathbf{v} = largest eigenvector by eigenvalue of $(\tilde{\mathbf{X}})$

$\forall i \in$ observed features:

$$w_i = \mathbf{v}^T \mathbf{x}_i$$

Calculate features to remove:

$$mad(X) = \text{median}_{x \in X} (|x - \text{median}(X)|)$$

$$lb(X, k) = \text{median}(X) - k * mad(X)$$

$$ub(X, k) = \text{median}(X) + k * mad(X)$$

$$I_{keep} = \{i \mid ub(\mathbf{W}_k, 2) > x > lb(\mathbf{W}_k, 2), \mathbf{w}_k^i \in \mathbf{W}_k\}$$

Update state with robust observations:

$$\mathbf{X}_{global}^{new} = \mathbf{X}_{global}$$

$$\mathbf{P}_{global}^{new} = \mathbf{P}_{global}$$

$\forall i \in I_{keep}$:

$$(\mathbf{X}_{global}^{new}, \mathbf{P}_{global}^{new}) = \text{UpdateGlobalCoordinates} \left(\begin{array}{l} \mathbf{X}_{global}^{new}, \mathbf{P}_{global}^{new}, \\ \langle \mathbf{X}_{feature}^j, \mathbf{X}_{feature}^k \rangle_i, \\ \langle \mathbf{P}_{feature}^j, \mathbf{P}_{feature}^k \rangle_i \end{array} \right)$$

Table 34 - Robust global coordinate update

7.2.4. Derivation of HAR-SLAM Updates

The key components of HAR-SLAM are the update algorithms, which improves the entire map through a series of Kalman-like updates. There are basic ripple updates, revisited landmark updates, and multiple robot landmark updates. Three main differences exist with HAR-SLAM updates when compared to the standard Kalman Filter updates. First, the ripple technique uses a sub-optimal gain in order to achieve the ripple effect. Instead of modifying both states given a new observation, the new observation is applied to one state and propagated to another. Second, the observation is not independent of the state, which is assumed in the Kalman Filter. However, the difference between the observation and state is independent of the state due to the Markovian property of the recursive filters [77][78]. Third, revisiting landmarks or updating multiple landmarks on the same local map cannot use the typical Kalman gain since it assumes independent and uncorrelated observations. However, if two landmarks are on a map, they are correlated and a different approach to the Kalman gain is required. This differing approach is optimal for correlated observations, but again not the typical gain used in the Kalman Filter.

The sub-optimal gain of the ripple technique has two effects that are required and verified in Table 36, Table 37 and Table 38. The selected Kalman gain applies the updated observed state directly to the source state in the given link, giving it the same state and covariance as the observed state. Second, if the source observation has the same covariance as the state, then no changes will be made to the source covariance, destination covariance, or cross-covariance. Table 36 and Table 37 show that the selected gain in the Link Update routine forces the source state and covariance to match

the observation state and covariance. Table 38 shows that having the same observation covariance as the original source covariance results in no changes to the destination covariance or cross correlation.

Given:

$$\mathbf{P} := \begin{bmatrix} \mathbf{P}_{source} & \mathbf{P}_{cross} \\ \mathbf{P}_{cross}^T & \mathbf{P}_{destination} \end{bmatrix} \quad \mathbf{x} := \begin{bmatrix} \mathbf{x}_{source} \\ \mathbf{x}_{destination} \end{bmatrix} \quad \mathbf{z} = \tilde{\mathbf{x}}_{source}^{new} \quad \mathbf{H} = [\mathbf{I} \quad \mathbf{0}] \quad \mathbf{R} = \tilde{\mathbf{P}}_{source}^{new}$$

Show:

$$\mathbf{K} = \begin{bmatrix} \mathbf{I} \\ \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} \end{bmatrix} \Rightarrow \mathbf{x}_{source}^{new} = \tilde{\mathbf{x}}_{source}^{new}$$

Derivation:

$$\begin{aligned} \mathbf{x}_{predict} &= \mathbf{x} \quad \mathbf{P}_{predict} = \mathbf{P} \\ \mathbf{z}_{predict} &= \mathbf{H}\mathbf{x}_{predict} = \mathbf{H}\mathbf{x} = \mathbf{x}_{source} \\ \tilde{\mathbf{y}} &= \mathbf{z} - \mathbf{z}_{predict} = \tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source} \\ \mathbf{x}^{new} &= \mathbf{x}_{predict} + \mathbf{K}\tilde{\mathbf{y}} = \mathbf{x} + \mathbf{K}(\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source}) \\ &= \begin{bmatrix} \mathbf{x}_{source} \\ \mathbf{x}_{destination} \end{bmatrix} + \begin{bmatrix} \mathbf{I} \\ \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} \end{bmatrix} (\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source}) \\ &= \begin{bmatrix} \mathbf{x}_{source} + (\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source}) \\ \mathbf{x}_{destination} + \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} (\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source}) \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathbf{x}}_{source}^{new} \\ \mathbf{x}_{destination} + \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} (\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source}) \end{bmatrix} \\ &\therefore \mathbf{x}_{source}^{new} = \tilde{\mathbf{x}}_{source}^{new} \end{aligned}$$

Table 36 - Verification that gain selection in link update results in source state matching observation state

Given:

$$\mathbf{P} := \begin{bmatrix} \mathbf{P}_{source} & \mathbf{P}_{cross} \\ \mathbf{P}_{cross}^T & \mathbf{P}_{destination} \end{bmatrix} \quad \mathbf{x} := \begin{bmatrix} \mathbf{x}_{source} \\ \mathbf{x}_{destination} \end{bmatrix} \quad \mathbf{z} = \tilde{\mathbf{x}}_{source}^{new} \quad \mathbf{H} = [\mathbf{I} \quad 0] \quad \mathbf{R} = \tilde{\mathbf{P}}_{source}^{new}$$

Show:

$$\mathbf{K} = \begin{bmatrix} \mathbf{I} \\ \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} \end{bmatrix} \Rightarrow \mathbf{P}_{source}^{new} = \tilde{\mathbf{P}}_{source}^{new}$$

Derivation:

$$\begin{aligned} \mathbf{P}^{new} &= \text{cov}(\mathbf{x}^{true} - \tilde{\mathbf{x}}^{new}) \\ &= \text{cov}(\mathbf{x}^{true} - (\mathbf{x} + \mathbf{K}(\tilde{\mathbf{x}}_{source}^{new} - \mathbf{H}\mathbf{x}))) \\ &= \text{cov}(\mathbf{x}^{true} - \mathbf{x} + \mathbf{K}(\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source})) \\ &= \text{cov}(\mathbf{x}^{true} - \mathbf{x}) + \text{cov}(\mathbf{K}(\tilde{\mathbf{x}}_{source}^{new} - \mathbf{x}_{source})) \\ &= \mathbf{P} + \mathbf{K}(\mathbf{R} - \mathbf{H}\mathbf{P}\mathbf{H}^T)\mathbf{K}^T \\ &= \mathbf{P} + \mathbf{K}(\tilde{\mathbf{P}}_{source}^{new} - \mathbf{P}_{source})\mathbf{K}^T \\ &= \begin{bmatrix} \mathbf{P}_{source} + (\tilde{\mathbf{P}}_{source}^{new} - \mathbf{P}_{source}) & \mathbf{P}_{cross} + (\tilde{\mathbf{P}}_{source}^{new} - \mathbf{P}_{source})(\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})^T \\ \mathbf{P}_{cross}^T + (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})(\tilde{\mathbf{P}}_{source}^{new} - \mathbf{P}_{source}) & \mathbf{P}_{destination} + \begin{pmatrix} (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})(\tilde{\mathbf{P}}_{source}^{new} \\ -\mathbf{P}_{source} \end{pmatrix} (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})^T \end{pmatrix} \end{bmatrix} \\ \mathbf{P}^{new} &= \begin{bmatrix} (\tilde{\mathbf{P}}_{source}^{new}) & (\tilde{\mathbf{P}}_{source}^{new} \mathbf{P}_{source}^{-1} \mathbf{P}_{cross}) \\ (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1} \tilde{\mathbf{P}}_{source}^{new}) & \begin{pmatrix} \mathbf{P}_{destination} + (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})(\tilde{\mathbf{P}}_{source}^{new}) \\ -\mathbf{P}_{source} \end{pmatrix} (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})^T \end{pmatrix} \\ \therefore \mathbf{P}_{source}^{new} &= \tilde{\mathbf{P}}_{source}^{new} \end{aligned}$$

Table 37 - Verification that gain selection in link update results in source covariance matching observation

Given:

$$\mathbf{P} := \begin{bmatrix} \mathbf{P}_{source} & \mathbf{P}_{cross} \\ \mathbf{P}_{cross}^T & \mathbf{P}_{destination} \end{bmatrix} \quad \mathbf{x} := \begin{bmatrix} \mathbf{x}_{source} \\ \mathbf{x}_{destination} \end{bmatrix} \quad \mathbf{z} = \tilde{\mathbf{x}}_{source}^{new} \quad \mathbf{H} = [\mathbf{I} \quad \mathbf{0}] \quad \mathbf{R} = \tilde{\mathbf{P}}_{source}^{new}$$

Show:

$$\mathbf{K} = \begin{bmatrix} \mathbf{I} \\ \mathbf{P}_{cross}^T (\mathbf{P}_{source})^{-1} \end{bmatrix}, \tilde{\mathbf{P}}_{source}^{new} = \mathbf{P}_{source} \Rightarrow \mathbf{P}^{new} = \mathbf{P}$$

Derivation:

Using last line from Table 37:

$$\begin{aligned} \mathbf{P}^{new} &= \begin{bmatrix} (\tilde{\mathbf{P}}_{source}^{new}) & (\tilde{\mathbf{P}}_{source}^{new} \mathbf{P}_{source}^{-1} \mathbf{P}_{cross}) \\ (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1} \tilde{\mathbf{P}}_{source}^{new}) & \left(\mathbf{P}_{destination} + (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1}) (\tilde{\mathbf{P}}_{source}^{new}) \right) \\ & \left(-\mathbf{P}_{source} \right) (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})^T \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{P}_{source}) & (\mathbf{P}_{source} \mathbf{P}_{source}^{-1} \mathbf{P}_{cross}) \\ (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1} \mathbf{P}_{source}) & \left(\mathbf{P}_{destination} + (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1}) (\mathbf{P}_{source}) \right) \\ & \left(-\mathbf{P}_{source} \right) (\mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1})^T \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{P}_{source}) & (\mathbf{P}_{cross}) \\ (\mathbf{P}_{cross}^T) & \left(\mathbf{P}_{destination} + \mathbf{P}_{cross}^T \mathbf{P}_{source}^{-1} (\mathbf{0}) \mathbf{P}_{source}^{-1} \mathbf{P}_{cross} \right) \end{bmatrix} \\ &= \mathbf{P} \\ \therefore \mathbf{P}^{new} &= \mathbf{P} \end{aligned}$$

Table 38 - Verification that using an observation covariance equal to the source covariance in link update has no effect on the source covariance, destination covariance, or cross-covariance

Upon observing a landmark again, a special update routine is called in order to accommodate the fact that the past landmark and current landmark are correlated through a chain of robot states. This correlation needs to be taken into consideration when calculating a gain that will propagate throughout the map. The proof that the correct gains are used when applying the correction to each landmark lies in the final values that result after the ripple update. If the gain is correct, then the two common landmarks will have the same position and have a difference of zero. Table 39 verifies that the gain choice in the update routine for revisited landmarks in Table 31 is valid.

Given:

Exists a connection from landmark A to pose 1 and several poses connect to pose N and pose N connects to landmark B

$$\text{Landmark A to pose 1: } \mathbf{P}_{A \leftrightarrow 1} := \begin{bmatrix} \mathbf{P}_A & \mathbf{P}_{A,1} \\ \mathbf{P}_{1,A} & \mathbf{P}_1 \end{bmatrix} \quad \mathbf{x}_{A \leftrightarrow 1} := \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_1 \end{bmatrix} \quad \mathbf{P}_{A,1} := \mathbf{P}_{1,A}^T$$

$$\text{Landmark B to pose N: } \mathbf{P}_{B \leftrightarrow n} := \begin{bmatrix} \mathbf{P}_B & \mathbf{P}_{B,n} \\ \mathbf{P}_{n,B} & \mathbf{P}_n \end{bmatrix} \quad \mathbf{x}_{B \leftrightarrow N} := \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_n \end{bmatrix} \quad \mathbf{P}_{B,n} := \mathbf{P}_{n,B}^T$$

$$\text{Pose i to pose i+1: } \mathbf{P}_{i \leftrightarrow i+1} := \begin{bmatrix} \mathbf{P}_i & \mathbf{P}_{i,i+1} \\ \mathbf{P}_{i+1,i} & \mathbf{P}_{i+1} \end{bmatrix} \quad \mathbf{x}_{i \leftrightarrow i+1} := \begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_{i+1} \end{bmatrix} \quad \mathbf{P}_{i,i+1} := \mathbf{P}_{i+1,i}^T$$

Show:

$$\left[\begin{array}{l} \mathbf{P}_{A \rightarrow B} := \mathbf{P}_{A,1} (\mathbf{P}_1)^{-1} \mathbf{P}_{1,2} (\mathbf{P}_2)^{-1} \mathbf{P}_{2,3} \dots \mathbf{P}_{n-1,n} (\mathbf{P}_n)^{-1} \mathbf{P}_{n,B} \\ \mathbf{S} := \mathbf{P}_A - \mathbf{P}_{A \rightarrow B} - \mathbf{P}_{A \rightarrow B}^T + \mathbf{P}_B \\ \mathbf{K}_A = \mathbf{P}_A \mathbf{S}^{-1} \quad \mathbf{K}_B = \mathbf{P}_B \mathbf{S}^{-1} \end{array} \right] \Rightarrow \hat{\mathbf{x}}_A = \hat{\mathbf{x}}_B$$

Derivation:

$$\vec{\mathbf{X}}_A = \mathbf{X}_A + \mathbf{K}_A (\mathbf{X}_B - \mathbf{X}_A) = \mathbf{X}_A + \mathbf{P}_A \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)$$

$$\vec{\mathbf{X}}_B = \mathbf{X}_B + \mathbf{K}_B (\mathbf{X}_A - \mathbf{X}_B) = \mathbf{X}_B + \mathbf{P}_B \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)$$

Next we ripple the update from both A and B through all the poses and onto one another

$$\mathbf{K}_{i \rightarrow j} = \mathbf{P}_{i,j}^T (\mathbf{P}_i)^{-1}$$

$$\begin{aligned} \vec{\mathbf{X}}_1 &= \mathbf{X}_1 + \mathbf{K}_{A \rightarrow 1} (\hat{\mathbf{X}}_A - \mathbf{X}_A) \\ &= \mathbf{X}_1 + \mathbf{P}_{A,1}^T (\mathbf{P}_A)^{-1} (\mathbf{P}_A \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)) \\ &= \mathbf{X}_1 + \mathbf{P}_{A,1}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \end{aligned}$$

$$\begin{aligned} \vec{\mathbf{X}}_2 &= \mathbf{X}_2 + \mathbf{K}_{1 \rightarrow 2} (\hat{\mathbf{X}}_1 - \mathbf{X}_1) \\ &= \mathbf{X}_2 + \mathbf{P}_{1,2}^T (\mathbf{P}_1)^{-1} (\mathbf{P}_{A,1}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)) \\ &= \mathbf{X}_2 + (\mathbf{P}_{A,1} (\mathbf{P}_1)^{-1} \mathbf{P}_{1,2})^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\ &= \mathbf{X}_2 + \mathbf{P}_{A \rightarrow 2}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \end{aligned}$$

$$\begin{aligned} \vec{\mathbf{X}}_3 &= \mathbf{X}_3 + \mathbf{K}_{2 \rightarrow 3} (\hat{\mathbf{X}}_2 - \mathbf{X}_2) \\ &= \mathbf{X}_3 + \mathbf{P}_{2,3}^T (\mathbf{P}_2)^{-1} (\mathbf{P}_{A \rightarrow 2}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)) \\ &= \mathbf{X}_3 + (\mathbf{P}_{A \rightarrow 2} (\mathbf{P}_2)^{-1} \mathbf{P}_{2,3})^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\ &= \mathbf{X}_3 + \mathbf{P}_{A \rightarrow 3}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \end{aligned}$$

⋮

$$\vec{\mathbf{X}}_i = \mathbf{X}_i + \mathbf{P}_{A \rightarrow i}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)$$

$$\vec{\mathbf{X}}_B = \mathbf{X}_B + \mathbf{P}_{A \rightarrow B}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)$$

$$\begin{aligned}
\tilde{\mathbf{X}}_n &= \mathbf{X}_n + \mathbf{K}_{B \rightarrow n} (\hat{\mathbf{X}}_B - \mathbf{X}_B) \\
&= \mathbf{X}_n + \mathbf{P}_{B,n}^T (\mathbf{P}_B)^{-1} (\mathbf{P}_B \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)) \\
&= \mathbf{X}_n + \mathbf{P}_{n,B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B) \\
\tilde{\mathbf{X}}_{n-1} &= \mathbf{X}_{n-1} + \mathbf{K}_{n \rightarrow n-1} (\hat{\mathbf{X}}_n - \mathbf{X}_n) \\
&= \mathbf{X}_{n-1} + \mathbf{P}_{n,n-1}^T (\mathbf{P}_n)^{-1} (\mathbf{P}_{n,B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)) \\
&= \mathbf{X}_{n-1} + (\mathbf{P}_{n-1,n} (\mathbf{P}_n)^{-1} \mathbf{P}_{n,B}) \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B) \\
&= \mathbf{X}_{n-1} + \mathbf{P}_{n-1 \rightarrow B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B) \\
&\vdots
\end{aligned}$$

$$\tilde{\mathbf{X}}_i = \mathbf{X}_i + \mathbf{P}_{i \rightarrow B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)$$

$$\tilde{\mathbf{X}}_A = \mathbf{X}_A + \mathbf{P}_{A \rightarrow B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B)$$

Ripples are additive so the two opposing ripples can simply be added at every pose, and at the end landmarks. To be more specific, the difference due to the ripple can be added.

$$\begin{aligned}
\hat{\mathbf{X}}_A &= \mathbf{X}_A + (\tilde{\mathbf{X}}_A - \mathbf{X}_A) + (\tilde{\mathbf{X}}_A - \mathbf{X}_A) \\
&= \mathbf{X}_A + \mathbf{P}_A \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) + \mathbf{P}_{A \rightarrow B} \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B) \\
&= \mathbf{X}_A + (\mathbf{P}_A - \mathbf{P}_{A \rightarrow B}) \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\
\hat{\mathbf{X}}_B &= \mathbf{X}_B + (\tilde{\mathbf{X}}_B - \mathbf{X}_B) + (\tilde{\mathbf{X}}_B - \mathbf{X}_B) \\
&= \mathbf{X}_B + \mathbf{P}_{A \rightarrow B}^T \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) + \mathbf{P}_B \mathbf{S}^{-1} (\mathbf{X}_A - \mathbf{X}_B) \\
&= \mathbf{X}_B + (\mathbf{P}_{A \rightarrow B}^T - \mathbf{P}_B) \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\
\hat{\mathbf{X}}_A - \hat{\mathbf{X}}_B &= (\mathbf{X}_A + (\mathbf{P}_A - \mathbf{P}_{A \rightarrow B}) \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)) - (\mathbf{X}_B + (\mathbf{P}_{A \rightarrow B}^T - \mathbf{P}_B) \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A)) \\
&= -(\mathbf{X}_B - \mathbf{X}_A) + (\mathbf{P}_A - \mathbf{P}_{A \rightarrow B} - \mathbf{P}_{A \rightarrow B}^T + \mathbf{P}_B) \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\
&= -(\mathbf{X}_B - \mathbf{X}_A) + \mathbf{S} \mathbf{S}^{-1} (\mathbf{X}_B - \mathbf{X}_A) \\
&= 0 \\
\therefore \hat{\mathbf{X}}_A &= \hat{\mathbf{X}}_B
\end{aligned}$$

Table 39 - Verification of gain selection in update revisited landmark

7.3. Advantage of HAR-SLAM

The primary advantage of HAR-SLAM is its low computational cost. The computation grows linearly with the number of states and landmarks, while in typical Kalman based SLAM algorithms computations grow quadratically, and in particle filters

computations grow exponentially. FastSLAM is the closest in computational cost, with the same linear growth with poses and number of landmarks; however, FastSLAM maintains several particles each with its own map compared to the single map in HAR-SLAM. Depending on how many particles FastSLAM has, FastSLAM could require more or less computation than HAR-SLAM.

Outside the scope of this dissertation, HAR-SLAM can have every link and state update in parallel. It will not necessarily get the exact optimal results per ripple, but the system should settle to the correct answer, and we can get a tremendous speed boost from the parallelization of HAR-SLAM. Parallel programming languages like CUDA, which runs on graphics cards can easily be used to speed up HAR-SLAM [79]. Showing the parallel version settles is also outside the scope of this dissertation. If the system does not settle, a practical addition would be a dampening coefficient (between 0 and 1) multiplied to each gain.

Another main advantage of HAR-SLAM is that it actively corrects the map and entire history of the robot's poses. FastSLAM only selects the best map, but does not correct the position of the robot. HAR-SLAM is similar to GraphSLAM and the Sparse Extended Information Filter (SIEF), which lend well to multiple robots and large map sizes. Those methods are known as "lazy" SLAM algorithms; they run in batch after the data is collected. HAR-SLAM is active, rippling changes to the entire map and to other robots.

A small advantage HAR-SLAM has over other methods is the ease and explicit definition of how to close-the-loop and merge multiple robot maps together. A simple

shortest path algorithm can find the chain of connecting poses between two landmarks, and update the entire system.

The last main advantage of HAR-SLAM is its robust nature. By utilizing the Robust Kalman Filter for both low-level sensor acquisition, and for joining several robots to a common map, HAR-SLAM is able to remove outliers, and recover from bad data associations. Kalman filters in general cannot recover from unexpected errors, and particle filters can only recover from unexpected errors if enough random particles exist to shift the system to the true state.

7.4. Disadvantages of HAR-SLAM

Every method has its shortcomings, and HAR-SLAM is no exception. Though HAR-SLAM is active, and utilizes Kalman-like filters, by the act of extracting poses and features into a chain makes the system sub-optimal. Though the required behaviors for the system were shown, without every cross correlation from every pose and feature to every other pose and feature, the gains and ripple are not exact but an approximation. Thus, HAR-SLAM cannot achieve the system-wide optimal gains that a Kalman-based SLAM could achieve. Both SEIF and GraphSLAM suffer from this as well, as sparsification and landmark removal become approximations of the system correlations, but not the actual correlations.

While HAR-SLAM is computationally fast, it still requires a considerable amount of time to execute. The ripple is not the slow part of the algorithm; rather the Forgetful SLAM portion is the slow section. Though Forgetful SLAM operates in linear time using

only the visible landmarks, the process has shown to be unable to keep up in a real time system due to the large number of landmarks that can be seen at once. Given the hardware used and experiments performed for this dissertation, the computational power required to process observations in real time was not feasible. It should be noted that Extended Kalman Filter SLAM and FastSLAM were also not feasible.

7.5. Computational Analysis

The key advantage HAR-SLAM has over other Kalman-based SLAM techniques is low computational complexity. HAR-SLAM only grows linearly over time, while both EKF-SLAM and GraphSLAM grow quadratically. FastSLAM is a particle filter based algorithm that has less computational complexity than HAR-SLAM; however, FastSLAM relies on having enough particles to correctly estimate the path and map. Since particles are not related to time, the computation time of FastSLAM does not grow over time, but instead remains constant. Particle resampling in FastSLAM does grow over time [27].

Table 40 shows the computational complexity of EKF-SLAM, GraphSLAM, HAR-SLAM, and FastSLAM. Estimates were not explicitly shown for FastSLAM particle resampling, which at best grows logarithmically with time [27]. The notation used for the computational complexity is big-O notation. The new landmarks observed at time t is n_t , on average it is noted as \tilde{n} . Reobserved landmarks at time t is r_t , on average it is noted as \tilde{r} . The dimension of the landmarks is d . Time is represented as t ,

which can be considered the number of iterations that have passed. For particle filters, k is the number of particles.

Algorithm	Computational Complexity
EKF-SLAM	(reobserved landmarks)*(update expanded state) $O\left((r_t)\left(\sum_{j=1}^t dn_j\right)^2\right) = O\left(\frac{\tilde{r}d^2\tilde{n}^2t^2}{2}\right) = O(\tilde{r}d^2\tilde{n}^2t^2) = O(t^2)$
GraphSLAM	(add observations to information matrix) + (reduce information matrix) + (solve information matrix) $O((r_t + n_t)d^2) + O\left(d(r_t + n_t)\sum_{i=1}^t d\right) + O\left(\left(\sum_{i=1}^t d\right)^2\right)$ $= O\left((r_t + n_t)d^2 + \frac{(r_t + n_t)d^3t}{2} + \frac{d^4t^2}{4}\right) = O(1 + t + t^2) = O(t^2)$
HAR-SLAM	(reobserved landmarks)*(update visible landmark state) + (add new landmarks) + (update past poses) + (update past landmarks) $O((r_t)(r_t)^2) + O(n_t(r_t)^2) + O\left(\sum_{i=1}^t d^2\right) + O\left(\sum_{j=1}^t n_j d^2\right)$ $= O\left(\tilde{r}^3 + \tilde{n}\tilde{r}^2 + \frac{d^2t}{2} + \frac{\tilde{n}d^2t}{2}\right) = O((\tilde{n}+1)d^2t) = O(t)$
FastSLAM	(number of particles)*(number of new and reobserved landmarks) *(update or create landmark) + (resample particles) $O(k(n_t + r_t)) * O(d^2) + O(k \log(t)) = O(kd^2(\tilde{r} + \tilde{n}) + k \log(t))$ $= O(\log(t))$

Table 40 - Comparison of SLAM computational complexity

Of the Kalman-based SLAM techniques, HAR-SLAM is less computationally expensive. Computation is linear in relation to time. Particle filter techniques are computationally superior to HAR-SLAM if the number of particles needed is less than $t/\log(t)$. In terms of storage space, Kalman-based techniques are typically quadratic, while FastSLAM is linear in terms of the number of landmarks. HAR-SLAM is linear compared to the number of landmarks, making it equivalent to FastSLAM.

7.6. Applications

HAR-SLAM was created with parallel processing, low computational cost, and multiple robots in mind. HAR-SLAM was designed to operate in real-time, given proper implementation and computational resources. Due to the scope of this dissertation and computational limits of available computers, all test results were processed after data collection, but simulated as if processed in real time. The end of Chapter 8 contains results relating to HAR-SLAM. The heart of Cooperative SLAM with Landmark Promotion (LP-SLAM) is HAR-SLAM using Forgetful SLAM with extra thresholds to reduce the global map size. HAR-SLAM is the theoretical component of LP-SLAM, enforcing that all operations are theoretically sound, with proven gains, and methods to handle all update scenarios. The next chapter goes into detail of the implementation of LP-SLAM, which includes spatial landmark storage, landmark matching, and landmark promotion.

CHAPTER 8: COOPERATIVE SLAM WITH LANDMARK PROMOTION

This chapter presents the more practical point of multiple robot HAR-SLAM, where theoretical algorithms are rephrased into implementable versions. Cooperative SLAM with Landmark Promotion (LP-SLAM) consists of HAR-SLAM using Forgetful SLAM as a low-level sensor fusion, and incorporating visual features using LK-SURF. Specific modules are outlined for storing landmarks, matching landmarks to determine loop-closure, and sharing landmarks amongst robots. Spatial storage techniques are presented to improve landmark lookup time. Landmark matching methods are presented to reduce the likelihood of false matches. Selecting a group of landmarks at a time, SURF recall methods are used as an initial matching technique followed by using a threshold on the Mahalanobis distances of homography transformations using select features [80]. A cooperative mapping module is presented that determines the global offset and rotation of each robot. Extra methods are presented on cooperative control, path planning, and an alternate network design using a central server.

8.1. Method Details

Cooperative Landmark Promotion SLAM is the combination of many elements presented in this dissertation. Figure 50 illustrates the different modules contained in Cooperative LK-SLAM. Initially, sensors are synchronized to match the image

acquisition frequency. After being synchronized, stereo images are processed with LK-SURF, which extract and identify visual features. Visual features along with the synchronized control and sensor data are sent to Forgetful SLAM. Forgetful SLAM maintains the current state. As poses and landmarks are removed from the current state, they are extracted and sent to the HAR-SLAM mapping module. This module maintains the map as a set of links (see Figure 49). Landmarks are sent to the loop-closing detector and to the global coordinate manager. Each of these modules maintains a spatial database of landmarks, which is illustrated by a common storage block in the diagram. The loop-closing detector compares incoming landmarks to the current database to find matches. Once a match has been made, the matched pair of landmarks is sent back to the HAR-SLAM mapping module. The global coordinate manager communicates with other robots. Only landmarks that pass a quality threshold are shared with other robots in order to limit network traffic (see Section 8.1.1.1). Cooperatively, the global coordinate manager determines the global coordinate transform of each robot, which results in an update to landmarks that were linked to other robots.

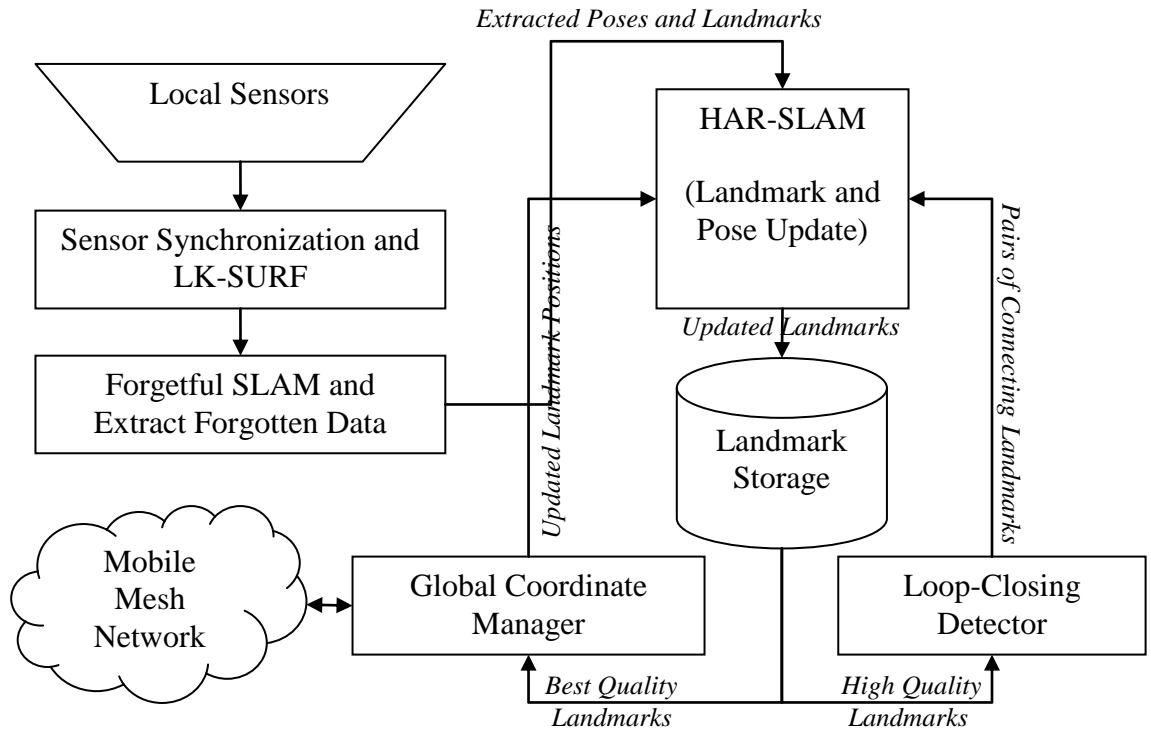


Figure 50 - Diagram of Cooperative LP-SLAM

The environmental data in LP-SLAM is the 3D visual landmarks. Using stereo cameras and LK-SURF, image features are projected into 3D space. Each feature has a descriptor from SURF, which consists of a vector of 64 floating-point numbers. As Forgetful SLAM runs, these features are correlated, filtered, and finally removed from Forgetful SLAM providing a location in the robot's local coordinate frame. In addition to a position, Forgetful SLAM provides an estimate of the covariance matrix associated with that map position, and a cross-covariance matrix linking that feature or map landmark to the last robot pose to see it. Thus, each landmark has five pieces of data, the position, the descriptor, a covariance matrix, a cross-covariance matrix, and the ID of the last pose to have seen that landmark.

8.1.1. Storing

In the HAR-SLAM mapping module, the storage of landmarks is done using a graph structure. Every pose state and covariance is attached to other poses and landmarks using references to their objects. Each reference is associated with a cross-covariance matrix in order to perform HAR-SLAM updates. Robot poses are linked and stored chronologically; this allows pose retrieval to occur in constant time. Landmarks are linked to only one state; landmark retrieval requires knowledge of the connecting pose. The graph structure of this module allows HAR-SLAM updates to occur in linear time with the number of elements in the graph.

In order to perform loop-closing, new landmarks need to be compared to the entire map of landmarks. Each feature or landmark has a descriptor from SURF. This descriptor is used for matching. Each landmark also has a position, a covariance matrix, a cross covariance matrix, and an ID of the pose that last saw the landmark. Matching not only involves descriptor comparisons, but position comparisons. An efficient approach for storing data is a hash-grid [81] [82]. Hash-grid storage uses a position based hash code to store landmarks in bins. A hash-grid can create bins dynamically as needed, thus storage space is not wasted with empty bins. With a hash-grid, if a location is known then lookup time is constant. Searching a space for landmarks is constant, but depends on the resolution of the bins, and the size of the search space.

The global coordinate manager module needs to access all landmarks regardless of location or connected pose. Storage for this module is a list of the features. The module shares the list with other robots, and in return receives lists of features from other

robots. Since there is no advantage to spatial storage or chronological storage, the only improvement that can be made to speed up performance and storage is to use fewer landmarks.

8.1.1.1. Landmark Promotion Metric

In order to limit the number of landmarks promoted to the global coordinate manager, a metric is created that reflects reliability. The cross covariance matrix between the landmark and pose is used in combination with the inverse of the landmark covariance matrix to determine how much a landmark can affect a pose (see Section 7.2.1: Standard Update). The gain used in HAR-SLAM is adapted as a metric of the landmark's certainty: $\max \text{Eigenvalue}(\mathbf{P}_{cross}^T (\mathbf{P}_{landmark})^{-1} \mathbf{P}_{cross})$. This metric is similar to a Mahalanobis distance [80], which is commonly used in statistics as a metric. If the certainty metric of the landmark is smaller than a set threshold, it can be discarded. Landmarks sent from Forgetful SLAM to the HAR-SLAM mapping module can be filtered using this metric. The same is true for landmarks sent to the loop-closing detector, and the global coordinate manager

8.1.2. Matching

Landmark matching is a critical process for both loop-closing and global coordinate determination. Since landmarks have SURF descriptors, a nearest neighbor ratio can be used to determine individual landmark matches. This is a typical feature matching technique used in computer vision [25]. The nearest neighbor ratio uses the

Euclidean distance between two SURF feature descriptors as a cost: $\sqrt{(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})}$. A match is determined if the lowest cost is less than a threshold fraction of the next lowest match using the same landmark. SURF was tested using various ratios; the best nearest neighbor ratio was 0.7, which achieved a recognition rate of 82.6% on a set of tested features[25].

Landmark Homography Calculation	
Matched Pair = $\langle \mathbf{x}_i, \tilde{\mathbf{x}}_i \rangle$	$\mathbf{x} = [x, y, z]^T$
$\mathbf{M} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3 & \tilde{\mathbf{x}}_4 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1}$	

Table 41 - Calculating landmark transformation

Matching SURF features requires a group of landmarks since the nearest neighbor matching technique is used. In addition to SURF matching, once a set of landmarks are matched, a distance metric can be used to verify the match. If a group of at least four landmarks is matched to another group then a homography calculation can be calculated. Table 41 shows the equation to calculate a homography transformation given a set of four matches. The transformation produces a rotation and translation that can be used to determine spatial fitness. Each match produces a spatial distance based on the Mahalanobis distance [80]. Table 42 shows the distance metric between two landmarks after the transformation is applied. The average distance of each match in a group is used as the group metric. If this average distance falls below a set threshold, the group can be accepted as successful matches. If a group contains more than four matches, homography calculation exists to best estimate the transformation; however, a successful technique used in image processing for homography calculations is random sample consensus (RANSAC) [22]. Selecting four matches at random from the group, the entire

group average distance can be calculated using the metric in Table 42. After a set number of random selections, the transformation with the lowest average distance is kept as the best transformation.

<p>Landmark Match Distance Function</p> <p>Matched Pair = $\langle \mathbf{x}_i, \tilde{\mathbf{x}}_i \rangle$ $\mathbf{x} = [x, y, z]^T$ $\mathbf{P}_i, \tilde{\mathbf{P}}_i = 3 \times 3$ covariance matrices</p> <p>Transformation Matrix: $\mathbf{M} = [\mathbf{R} \quad \mathbf{t}] = 3 \times 4$ matrix of a rotation and translation</p> <p>Cost = $\sqrt{(\tilde{\mathbf{x}}_i - \mathbf{R}\mathbf{x}_i - \mathbf{t})^T (\mathbf{R}\tilde{\mathbf{P}}_i\mathbf{R}^T + \mathbf{P}_i)^{-1} (\tilde{\mathbf{x}}_i - \mathbf{R}\mathbf{x}_i - \mathbf{t})}$</p>
--

Table 42 - Calculation of landmark fitness

Matching landmarks can occur at any time thanks to the nature of HAR-SLAM. Thus, we can consider landmark matching a background process that occurs at any interval. Incorrect landmark matching is avoided by comparing groups of landmarks to other groups of landmarks. Comparing two groups requires that the SURF descriptors match, and that an affine transformation can map one group's coordinates to the other group's coordinates. Setting overly strict thresholds reduces the risk of outliers entering the HAR-SLAM mapping routine.

When matching a group of landmarks, the entire map of landmarks should be compared to the current group to determine if a loop has been closed. However, this process can be sped up by limiting comparisons to a certain subset of landmarks, which is not optimal, but practical. Each landmark has a covariance matrix that expresses the uncertainty in the landmark's position estimate. The size of the search space around each landmark can be defined as a function of the uncertainty by limiting the lookup space relative to the covariance matrix. In practice, this can be done by finding the eigenvectors of the covariance matrix, then finding the rotated bounding box that bounds

a set ratio of each eigenvector in both positive and negative directions to form the region. Since the landmarks are stored in a hash-grid, each landmark has a very limited lookup space, and the search for potential matches requires very few operations.

8.1.3. Map merging

Map merging is a difficult process in most SLAM algorithms. SEIF has shown to lend itself well to the multiple robot case, but only as an aftermath in batch mode [27]. HAR-SLAM allows multiple robots to join maps actively whenever an outside process decides what landmarks to join. In order to join maps, the algorithm needs to decide when features are in common, and which features to select.

The global coordinate manager is the external module to HAR-SLAM that decides which features are shared with other robots, and calculates the global coordinate transformation. The promotion metric (Section 8.1.1.1) can be used to limit the landmarks the global coordinate manager receives. The module shares landmarks with all robots in the mesh network, and receives landmarks from other robots. The array of landmarks from each robot is compared to every other robot. The same landmark matching technique that is used in loop-closing is used to determine matches between robots. Though landmarks are in different coordinate frames, the homography transformation still works. A strict threshold on the average distance metric should be used to limit the chance of false matches.

The global coordinate manager cooperates with other robots by dividing work. Each robot compares its own set of landmarks to all other robots. There is a level of

redundancy by a factor of two, but this redundancy can be used to validate a match. Each match is shared amongst all robots. After every robot completes the matching phase, the global coordinate update filter can be used (see Section 7.2.3: Multiple Robot Updates). The method is deterministic, so each robot can calculate the global coordinate transformation locally using the common list of matched landmarks each robot received. The filter uses a subset of matched landmarks. The subset of matched landmarks involving the local robot is updated based on HAR-SLAM update equations (see Section 7.2.3). The updated landmarks are sent to the HAR-SLAM mapping module, as indicated in Figure 50. The global coordinate manager operates independently of the HAR-SLAM mapping module, and can perform updates at any time.

8.2. Related Topics

The material covered in this section is relevant to Cooperative LP-SLAM; however, the details are not investigated in this dissertation. Cooperative control is important for autonomous mapping of buildings. This section includes a brief discussion of cooperative control. LP-SLAM and HAR-SLAM have an interesting advantage based on map storage. By using a set of linked poses, the path connecting two regions is always known. Thus, path finding is an interesting covered briefly in this section. Finally, a brief mention of client-server based global mapping is discussed.

8.2.1. Cooperative Control

A few approaches to cooperative control are presented earlier (see Section 2.3: Cooperative Control and Coverage). A relatively simple and effective approach for increasing the probability of full coverage is the use of pseudo-potential fields. Each robot generates a potential field on the global map. This information was explained in detail in section 2.3.3 on page 50. For use in map exploration, the potential of explored areas can be increased to have robots to spread away from explored regions causing them to map an entire area. The potential field presented in the earlier section included costs to maintain network connectivity and prevent collisions between robots. Given that the robots are producing a map, the potential could be increased for detected walls and obstacles, which combines obstacle avoidance, exploration, and network connectivity. There is no guarantee of optimality, but the solution is simple and effective.

Other control methods, such as tree graphs, optimal exploration using gradient descent, or simple cell coverage are applicable. Work from section 2.3.5 on page 54 shows methods to explore a known region optimally. The same authors present work on exploration using similar methods.

8.2.2. Path finding

LP-SLAM has a unique advantage in map formation. Every landmark and data sample is associated with a robot location; this means that every point of the map has an associated pose and thus a chain of poses that connect any two points on the map. Overlapping paths and multiple robot paths would take a little extra work to create

intersection nodes, but the entire map is already a graph of poses. Simple path planning algorithms can run on top of this connected graph in order to find shortest paths.

This application allows robots not only to map a region, but return to any position in the map. Navigation becomes much simpler when a known path exists, only obstacle avoidance needs to be considered along a set trajectory. This idea can be expanded to have special sensors pick up or interpret labels, allowing the robot to know where a particular label is associated on the map, and allowing the robot to return to that position. Such sensor could be speech recognition engines that interpret voice commands, or optical character recognition that could read door numbers.

8.2.3. Central Server Map Manager

Instead of having the global coordinate manager cooperatively determine the global coordinate transformation, a server could be used. Instead of updating all robots, update only the server. The server would be responsible for the global coordinate system. It would perform the global coordinate update filter, report which matches were used in the filter, and transmit the newly updated coordinate system to each robot individually. This method would reduce network traffic by a factor linearly dependent on the number of robots; however, this increases the workload on a single machine. The increase in workload is linearly related to the number of robots. One point of performance increase is that only one machine would perform the global coordinate update filter; though, this allows for a single point of weakness in the network.

8.3. Examples

This section presents some examples that show maps LP-SLAM and HAR-SLAM produce. The examples in this section used LK-SURF to track visual features, filtered observed data using Forgetful SLAM, updated past map data using HAR-SLAM mapping, then finally detected and applied loop-closing.

Figure 51 compares the path generated by Forgetful SLAM and the path generated by HAR-SLAM. HAR-SLAM continuously modifies past elements using the rippling update. The updates appear to reduce the effect of outliers (notice the right side of the loop near coordinate (6, 1)). Loop-closing performed the final correction to the path and map. The Forgetful SLAM routine took 4.95 minutes to operate on a 2 minute path. All the HAR-SLAM rippling updates took 53.5 seconds. The loop-closing update only took 0.09 seconds. The total computation time of LP-SLAM was 5.84 minutes. EKF-SLAM took 16.98 hours, and did not include loop-closing (see Figure 47 in Section 6.2).

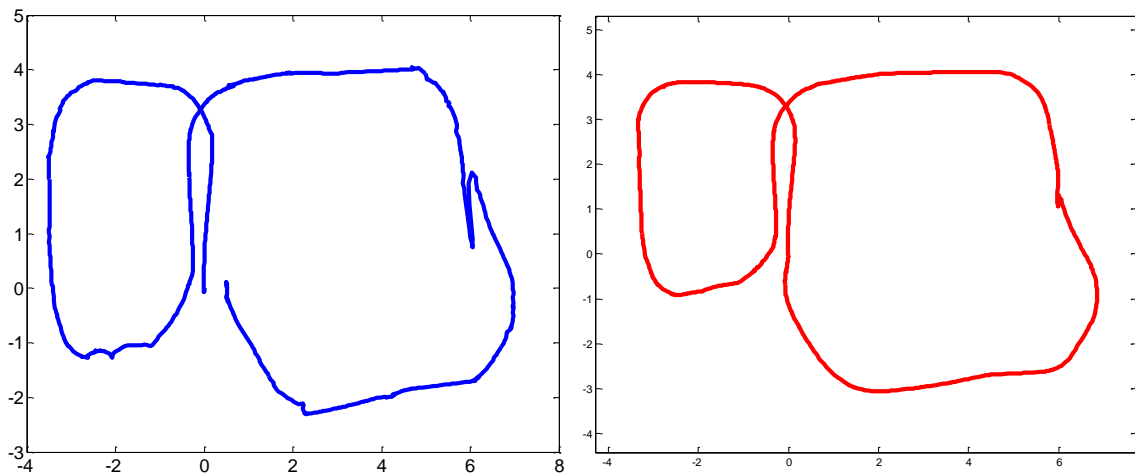


Figure 51 - Figure-eight path; Left: Forgetful SLAM path; Right: HAR-SLAM updated path

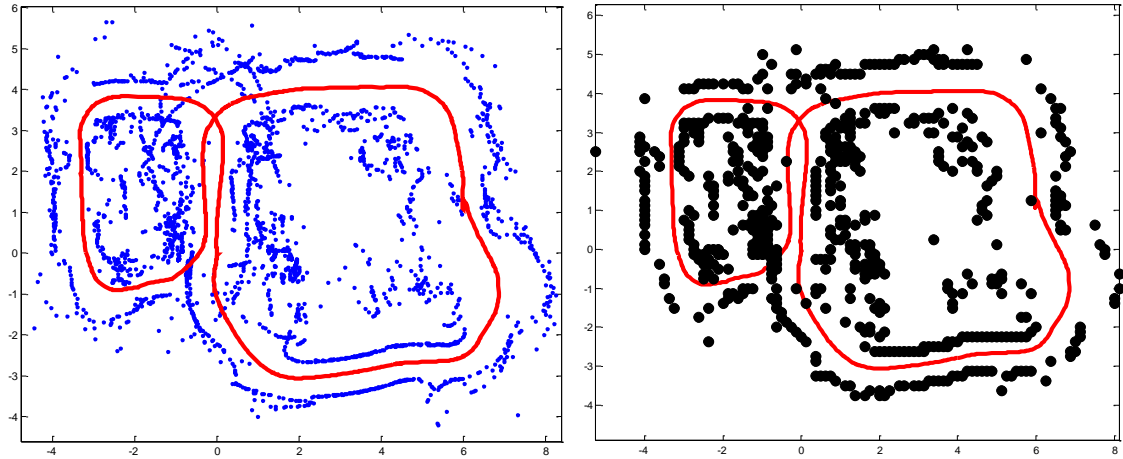


Figure 53 - Sonar produced from HAR-SLAM path; Left: Raw sonar values; Right: Filtered sonar values

Figure 54 shows the hybrid map of the sonar map from Figure 53 and visual landmarks from Figure 52. Some visual features align with detected obstacles. Many visual features exist randomly around the map; this is because features could be ceiling lights, floor tiles, textures on the wall, or any object in the room.

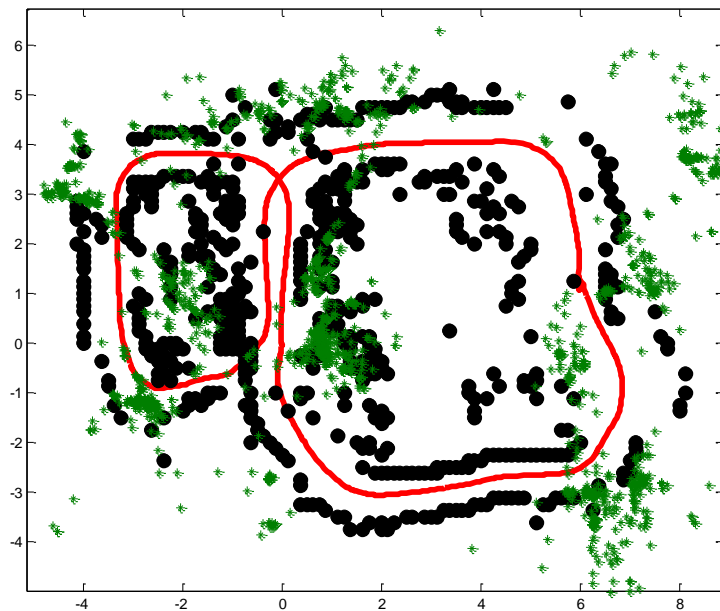


Figure 54 - Hybrid sonar and visual landmark map on figure-eight path

Figure 55 shows two other maps generated using HAR-SLAM. The map on the left is the same area as the smaller loop on the left in Figure 54; the right map is the same as the larger loop on the right in Figure 54. Some of the visual landmarks appear to be in the same location as in Figure 54, which would allow for map merging.

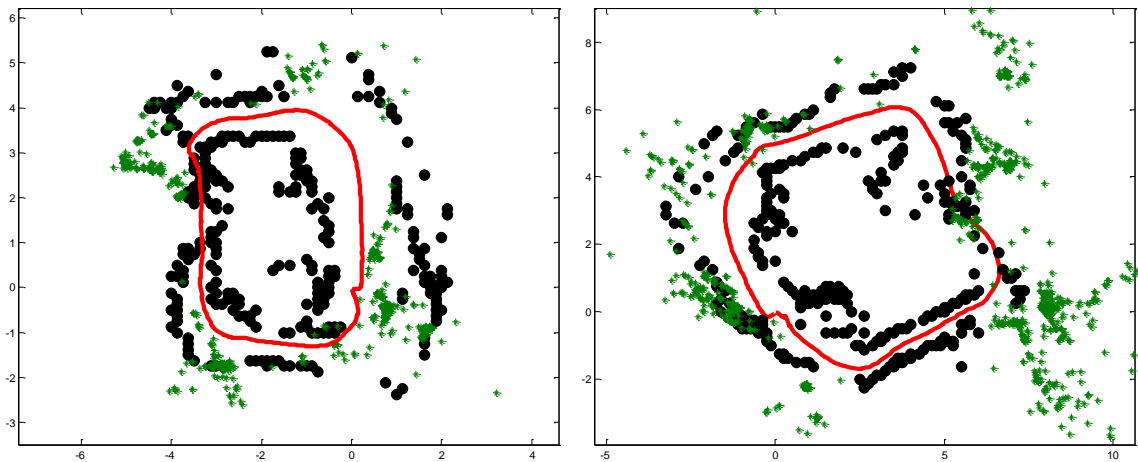


Figure 55 - Hybrid sonar and visual landmark map on short counterclockwise loop (left) and long clockwise loop (right)

Figure 56 shows a scale CAD drawing of the test location. The drawing includes the approximate location of furniture in the environment. On the right, the figure-eight path is drawn on top with ranging to obstacles and visual landmarks. Given the CAD drawing as a reference, visual landmarks appear to be located around furniture, walls, and doorframes. Obstacle ranging shows the bounds of walls and tables.

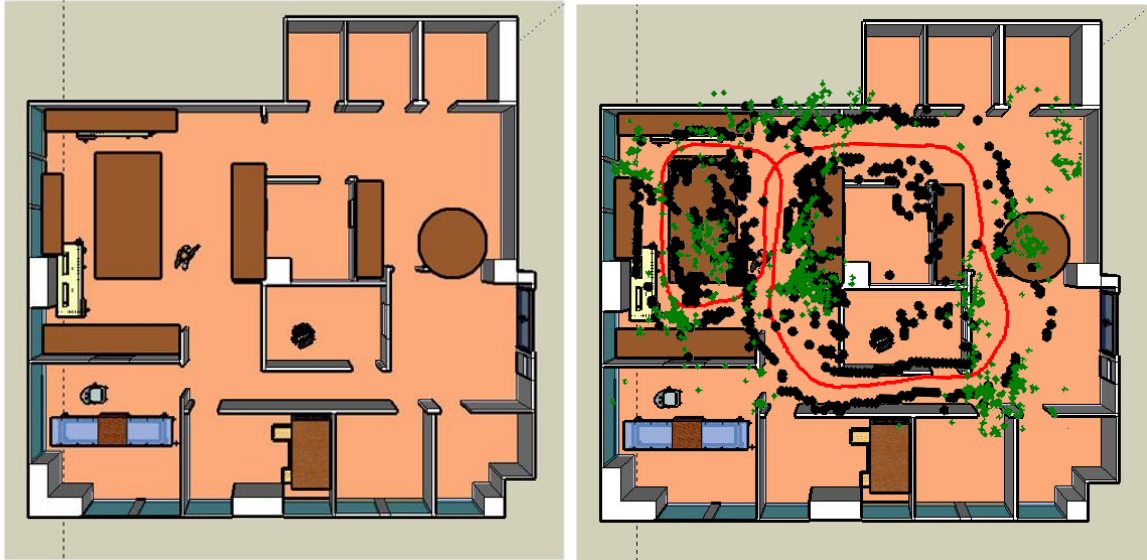


Figure 56 - CAD drawing of the test location; Left: layout with approximate location of furniture; Right: HAR-SLAM map and path drawn on top of the CAD drawing

As an example, the three separate maps, generated using HAR-SLAM, are merged by calculating the global coordinate transform. First, to reduce computational complexity, features are filtered using the promotion metric (see Section 8.1.1.1: Landmark Promotion Metric). Figure 57 shows the three example paths with features that were selected for promotion circled in red. For these examples, a threshold of 0.002 was used to promote landmarks. A histogram of the landmark strengths, as calculated by the landmark promotion metric, is shown in Figure 58 . The histogram contains data for the figure-eight path. Figure 59 shows the merged map of visual landmarks from all three paths. Many of the selected landmarks are brought within close range of each other. Figure 60 shows the sonar map, which shows how well the walls and obstacles align for this given example.

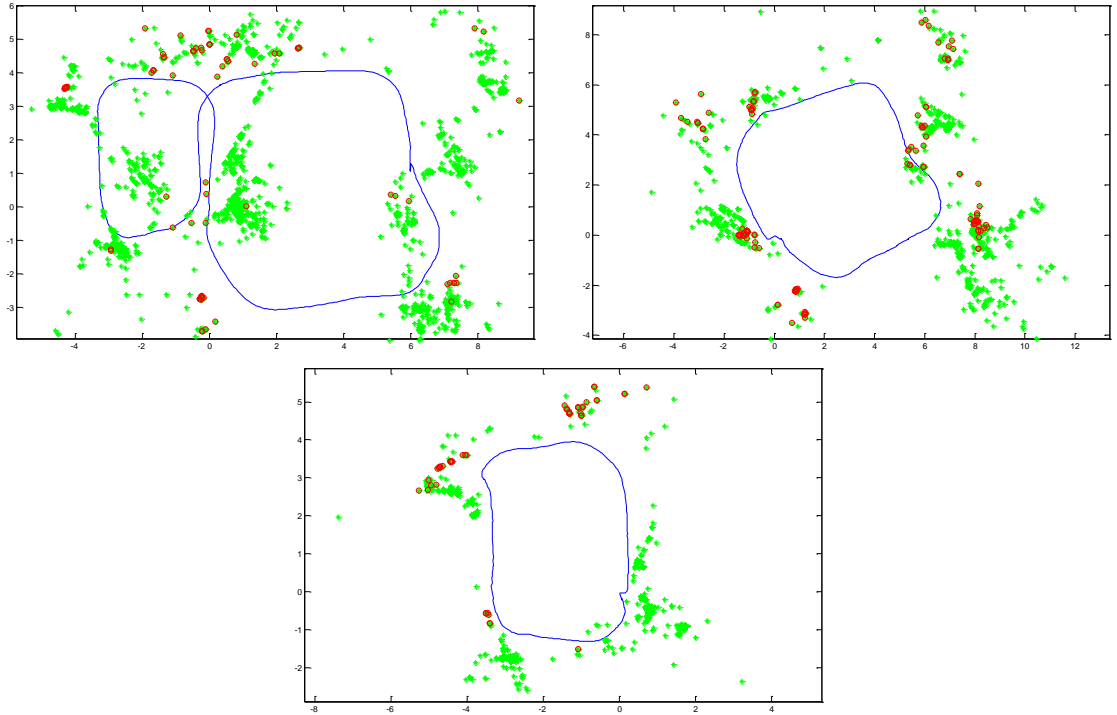


Figure 57 - Paths with marked features for promotion; Top-Left: Figure-eight path; Top-Right: Long clockwise path; Bottom: Short counterclockwise loop

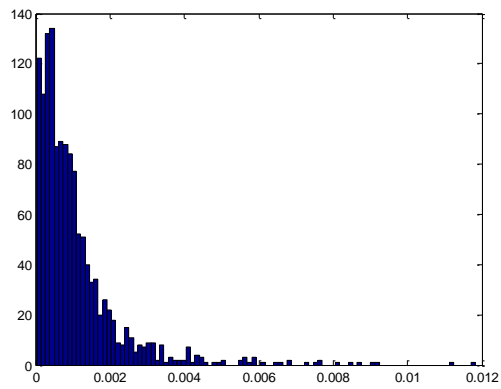


Figure 58 - Histogram of landmark strength, calculated by the landmark promotion metric for the figure-eight path

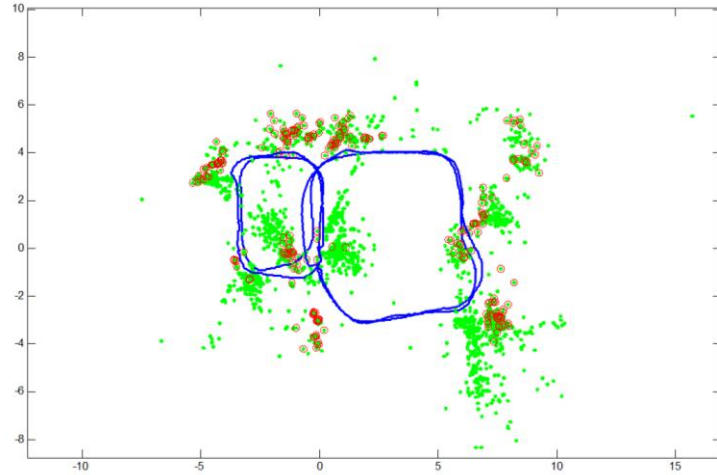


Figure 59 - Visual landmark maps generated by Cooperative LP-SLAM using Figure-eight path, long clockwise path, and short counterclockwise path

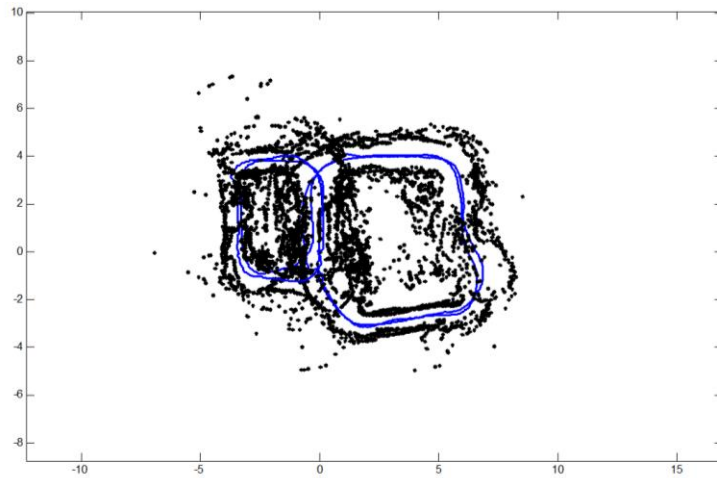


Figure 60 - Sonar maps generated by Cooperative LP-SLAM using Figure-eight path, long clockwise path, and short counterclockwise path

CHAPTER 9: HARDWARE AND DESIGN

9.1. Building the Robot

The test platform used for this dissertation was a custom designed and built robot. The chassis of the robot is a Tamiya TXT-1 remote control truck. The upper platform is custom made, as are the sensor acquisition boards, the optical inertial navigation unit, wheel encoders, and the mounting brackets. Figure 61 shows several views of the fully assembled test platform. The platform consists of two Firefly MV cameras from Point Grey mounted in front; under the cameras is the TRX INU with glowing lights indicating the unit is on. Those three components make up the Optical INU, which is mounted on the front of the robot. In the center is an AOpen miniPC, which is a miniature desktop with an Intel Core 2 Duo 2.0GHz process and 2.0 GB of RAM. A high-powered Buffalo router is on the opposite side of the robot from the cameras. The blue packs on the sides of the robot are lithium-ion batteries that are connected to a DC voltage converter located under the router. The batteries power the PC, the router, and the USB hub. Around the outside of the custom mounting platform are eight ultrasonic ping sensors that are typically used for obstacle avoidance. Not clearly shown in this picture are two sensor acquisition boards, one used to acquire ultrasonic ping sensor data, the other to collect wheel encoder data. The wheel encoders are located inside the hub of each wheel and are not clearly shown. Finally, a Parallax servo controller is located under the custom platform to interface from the computer to the motor controller and steering servo.

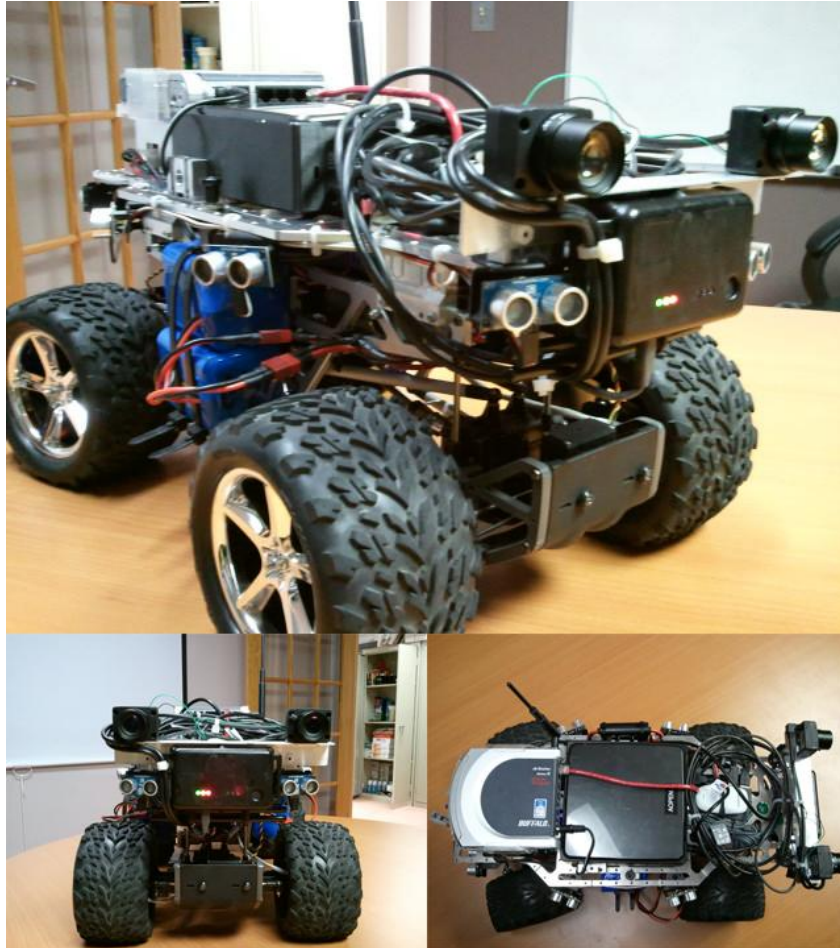


Figure 61 - View of the test platform

The metal platform on top of the Tamiya chassis was custom made and designed in the Autonomous Systems Laboratory (ASL). Figure 62 shows the design of top platform (in yellow), the connecting brackets (in blue and red), and the accessory mounts (in green). The platform has proved to very useful as most sensors, electronics, and accessories are mounted to the main platform.

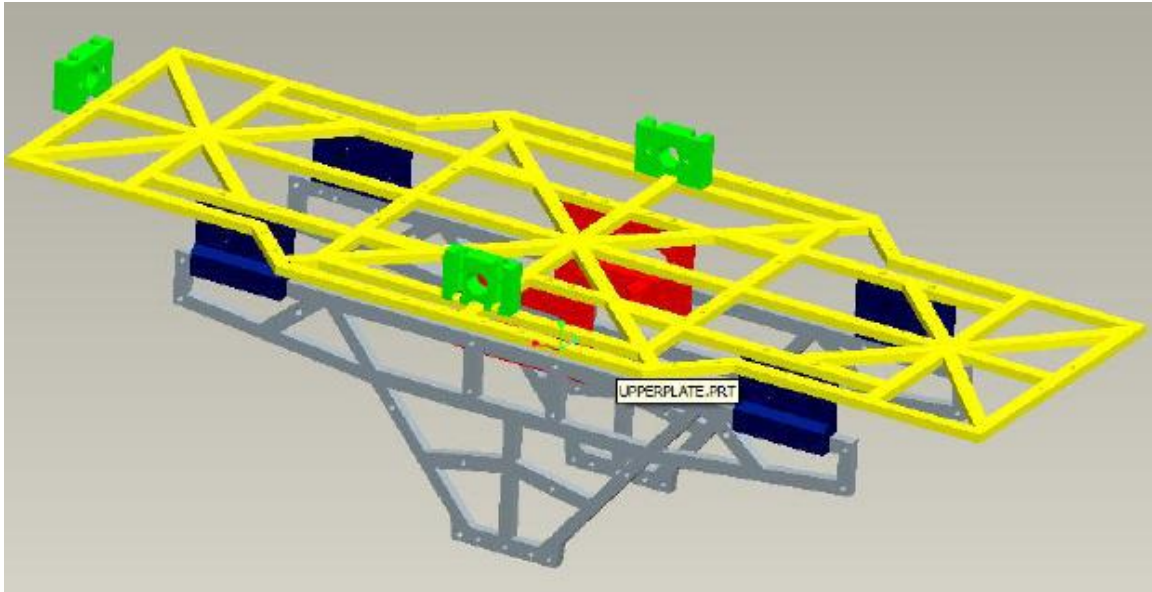


Figure 62 - CAD design of custom chassis modifications [83]

The entire list of parts to build the test platform is shown in Table 43. The total cost of parts is estimated to be between \$2,000 and \$3,000, which is half target cost. This platform meets the goal of an inexpensive mobile sensing unit with high computational power.

Quantity	Component
1	Tamiya TXT-1 Monster Truck 4WD
1	Set of Custom ASL Platform and Brackets
1	AOpen Core 2 Duo miniPC (Windows XP Pro)
1	Buffalo WHR-HP-G54 Wireless Router
1	CarNetix CNX-1900 DC-DC Regulator
1	Novak Goat Crawler ESC
1	Parallax Servo Controller
2	ASL Sensor Boards
2	Point Grey Firefly MV Cameras
4	11.1V 6600mAh Li-Ion Battery Pack
2	8.4V 4200mAh NiMH Battery Pack
4	US Digital E4P Miniature Optical Encoder Kit
8	Parallax Ultrasonic Ping Sensors
2	Futaba HS-5645MG digital high torque servo
1	TRX Systems Inertial Navigation Unit (INU)
1	Wireless Xbox Controller and USB Receiver

Table 43 - Test platform components

9.1.1. Sensor Board

At the Autonomous Systems Lab (ASL), a sensor acquisition board was designed. It proved to be of great use for this dissertation as a sensor acquisition board for the ultrasonic ping sensors and for the wheel encoders. The design is flexible enough that it was repurposed for independent projects as a motor controller, as a voltmeter, and a full array of digital, analog, and pulse width modulated sensors. A diagram of the sensor board, as well as a schematic of the layout, and the actual board itself is shown in Figure 63. The pin layout allows many different sensors to plug into the board directly. The board has a USB interface to the PC, and communicates over a serial protocol.

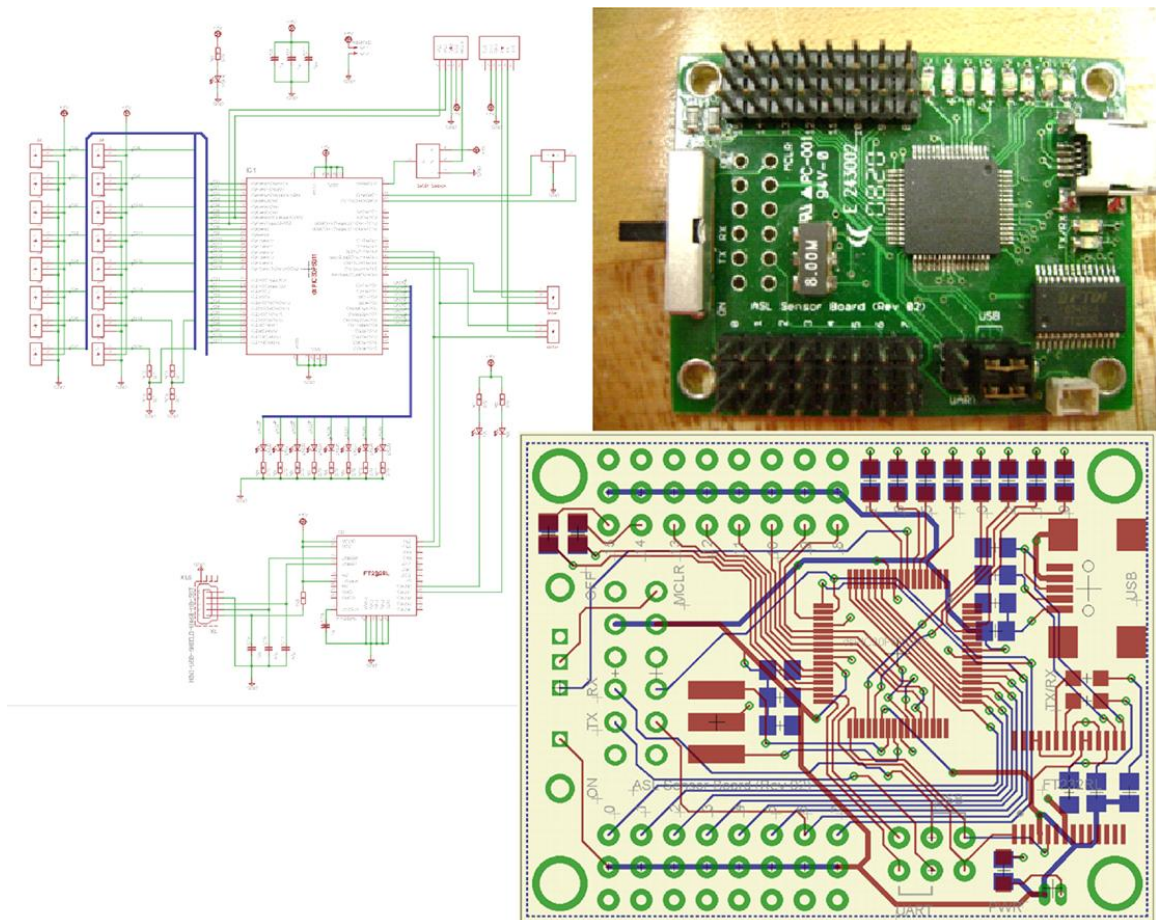


Figure 63 - Diagram, schematic, and view of the sensor acquisition board [83]

The ultrasonic ping sensors are visible in the main image of the test platform, but the encoders are hidden inside the wheel hubs. Figure 64 shows the inside of the wheel hub, revealing the small encoder placed on the shaft. In the same figure, the individual parts of the encoder are shown, including the protective enclosure, the sensor board, and the marked disc. With approximately 1000 marks per rotation, the encoder has a resolution of 0.36 degrees.

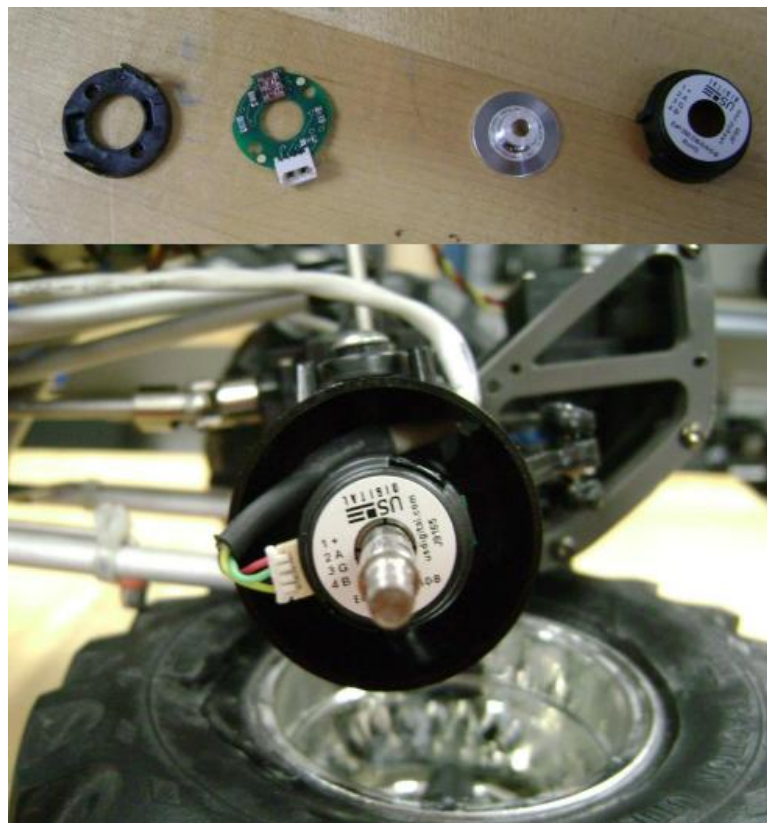


Figure 64 - Encoder parts and assembled encoder [83]

9.1.2. Optical INU

The Optical INU is a combination of Point Grey Firefly MV cameras and the TRX Systems Inertial Navigation Unit. The Firefly MV cameras are inexpensive machine vision cameras. Like most machine vision cameras, the lenses are

interchangeable; this allows future work of comparing different lenses against Optical INU performance. Point Grey provides a .NET library to make interfacing to the camera quite simple. The TRX INU is a custom-made tracking device by TRX Systems [9][15] [84] [85]. The device was designed to track humans in GPS-denied environments. It is typically worn on a belt around the waist. The device can also stream calibrated sensor data via USB. TRX developed a .NET library to interface with the device as well, making the data acquisition quite easy. Figure 65 shows a TRX INU with a belt attached and the Firefly MV cameras, forming the Optical INU.



Figure 65 - Close view of the TRX INU with Point Grey cameras, the Optical INU

9.1.3. Computing Platform

Each robot is equipped with a full PC. The PC, manufactured by AOpen, is a complete Windows XP machine with a Core 2 Duo processor at 2 GHz. Resembling a Mac Mini, the PC collects all sensor data and controls the robot's movement. An Xbox

controller and USB receiver are attached to the robot to manually control its movement, and override an algorithm if the robot appears to be veering off course. This same platform is used for many projects in the Autonomous Systems Lab. The PC does not contain a Graphics Processing Unit (GPU) that can run CUDA or OpenCL. Due to the lack of a GPU, no parallel processing techniques can be explored using this computer. However, it is easy to replace the PC with a newer one since components are connected via USB.

9.1.4. *Mobile Mesh Network*

The robots use Buffalo WHR-HP-G54 Wireless-G routers, which were chosen for their long-range transmission. The routers were reprogrammed with the Freifunk firmware [86], which is an open-source project that enables advanced algorithms and features on common routers. The firmware supports a mesh networking protocol known as Optimized Link State Routing (OLSR). The system and OLSR settings were fully programmed and configured successfully. Figure 66 shows sample mesh networks formed with the OLSR algorithm. The visual representation was created with the Freifunk OLSR mesh networking visualizer.

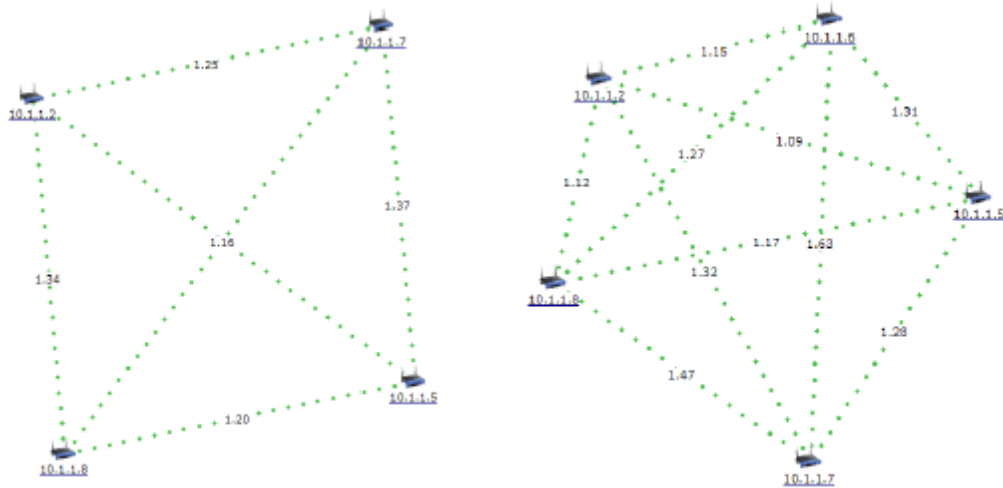


Figure 66 - Freifunk OLSR mesh network visualizations [86] [87]

9.2. Programming the Robot

A software framework, the ASL Framework, was created at the Autonomous Systems Lab. This framework was written in C# and works with all Windows Platforms. Other frameworks were considered, such as Microsoft Robotics Studio [80], AForge.NET [88], and Roborealm [89]. Only recently did Microsoft Robotics Studio supports multiple robots [90]. Microsoft Robotics Studio has several criticisms associated with development complexity; its use was abandoned by several research groups including ASL[91]. AForge.Net and Roborealm do not support multiple robots. During testing, they proved to be unreliable and overall performance was slow compared to the ASL Framework.

The ASL Framework makes use of several third-party libraries. Within this framework is an image processing suite based on Intel's OpenCV library [22]. For use in C#, a wrapper library called EMGU is used; it simply calls the native C++ OpenCV routines in C#. Image processing capabilities include color filters, edge detectors,

blurring, corner detection with Harris corners [23], face tracking, and feature tracking with Lucas-Kanade Optic Flow [24].

This ASL Framework supports various hardware platforms, such as the Tamiya based platform used in the Autonomous Systems Lab. Additional platforms are supported, such as UMD's ENEE408I class robots, certain helicopters, the iCreate from iRobot, and several others. The framework is flexible enough to allow any platform to be integrated into the system. The framework includes networking capabilities, which allow the robots to behave as client-server models or as peer-to-peer networks. The framework's network implementation allows the robots to disconnect and reconnect at any given time—this allows the system to be as robust as possible with any network hardware. Given that the current platform implements mobile mesh networking, the system may break and reconnect at any given time without adverse effects.

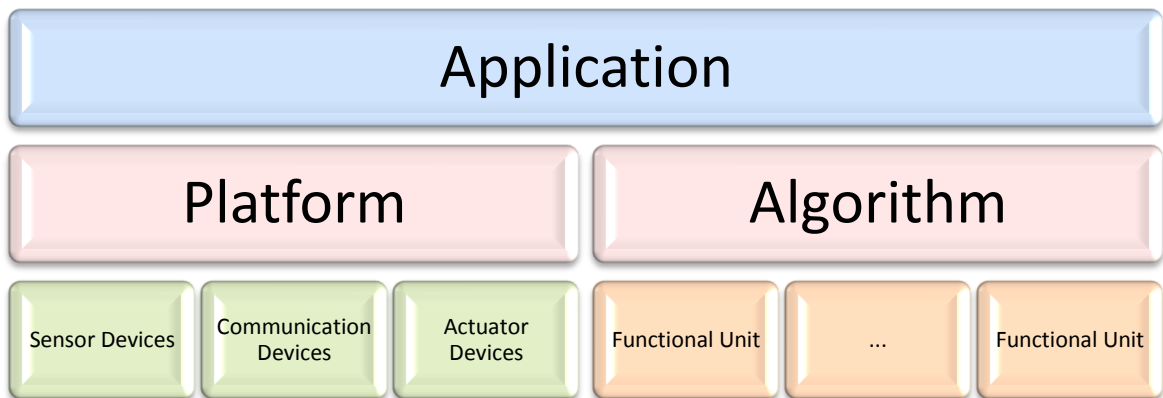


Figure 67 - ASL Framework hierarchy

Figure 67 shows the ASL Framework hierarchy. The executable application controls both a platform and an algorithm. A platform is defined as collection of sensors, actuators, and communication devices. For example, a camera and server network configuration could be described as a platform and used with various algorithms. The

functional units perform core functions, such as feature detection, image processing, or perhaps a Kalman Filter. An algorithm is simply a collection of functional units in a particular order. They can be setup in parallel, in series, and even in cycles; however, cycles need to be carefully dealt with in code to prevent infinite loops.

The entire system works as a flow as shown in Figure 68. Data is collected from the sensors and from inbound communication from other robots. The data is transferred from the platform and into the algorithm. The algorithm then links this data into a chain of functional units. Afterwards, the chain returns outbound communication (the network and actuator information), and the robot moves or acts.

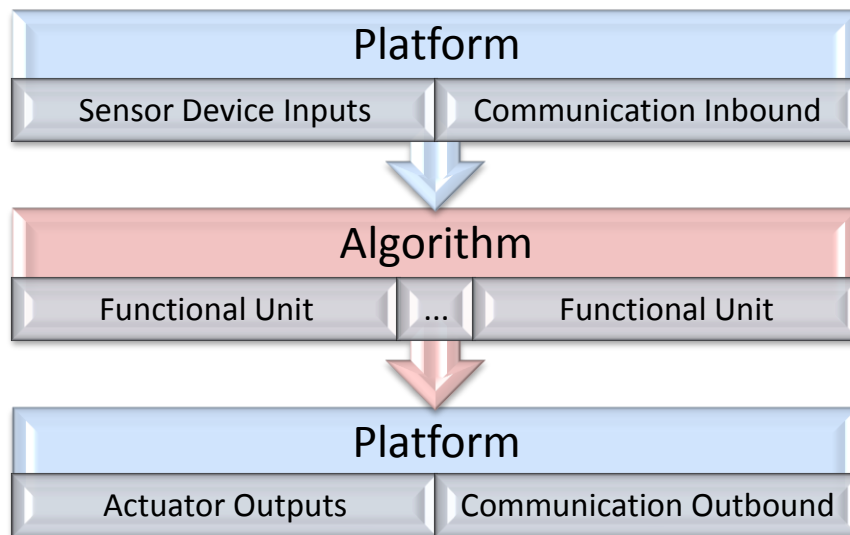


Figure 68 - ASL Framework data flow

It is important to note that the framework is simply a way to organize algorithms and easily interface with hardware. Each component of the framework can be customized for any application. Our framework has been tested with UMD's ENEE408I class. It was used in independent research projects, and used in dissertations by graduate students[83][87]. Students used this framework to program their own robots

successfully. By using the ASL Framework, their robots could cooperatively play a simplified version of soccer, perform a race, or map a hallway.

A few sample screenshots of the ASL Application are shown in Figure 69. The ASL Application is the main program that selects the platform and algorithm, and actually initializes and starts the system. This application could be modified for command line versions or remote operation versions, but for simplicity, it is just a Windows application. These particular examples show a virtual platform being used; a virtual platform is a special platform that replays recorded data providing a fake robot. The virtual platform allows algorithms to be tested against the exact same data. The examples show the various displays the framework contains, with no end in options, from 3D rendering, to 2D plots, to stereo camera views.

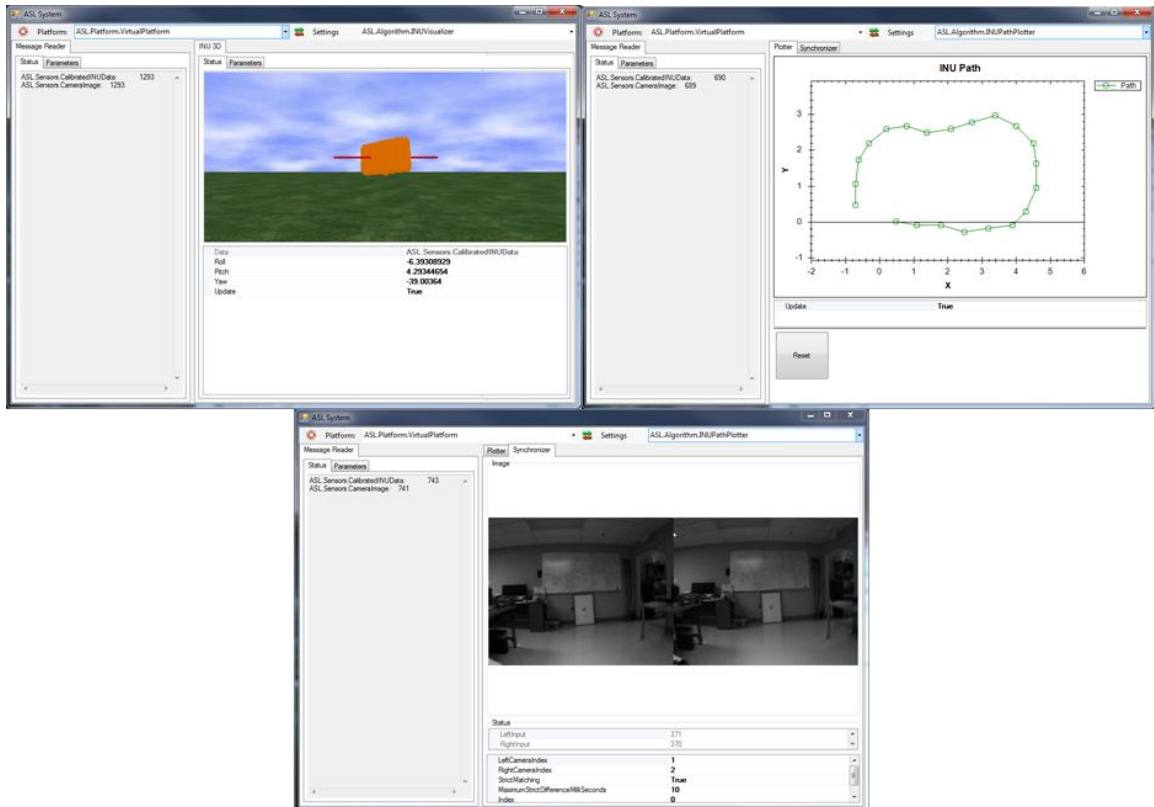


Figure 69 - Sample screenshots of the ASL Application

9.2.1. Event Architecture

An event-based architecture is a key component of the ASL Framework. In .NET languages like C#, event driven patterns are very fast. When an event is fired, functions listening to that event are called. In the ASL Framework, these events are all fired asynchronously, allowing multiple operations to perform in parallel across multiple cores. Typically, the ASL Framework can occupy all cores of a processor if a task requires the computing power. The framework takes care of all parallel processing issues by internally providing data locks and semaphores, thus there are no deadlock issues, or race conditions. Furthermore, if a functional unit becomes a bottleneck, the framework is designed to drop data in order to prevent the system from lagging behind.

9.2.2. Platform

The platform component of the ASL Framework is a collection of devices. A device interfaces either to hardware or to a network. Devices that interface to hardware have only two primary functions, collect sensor data and implement actuator data. Collecting sensor data results in listening to a port, polling for data, or directly interfacing using a third-party library. Once a sample of data has been collected, the sample is transmitted to the algorithm portion of the framework using the event architecture. Implementing actuator data involves receiving actuator packets that instruct a particular type of device to perform an action. This action is then carried out by interfacing to an external device, or sending instructions to a third-party library.

Communication based devices handle incoming and outgoing data to and from other robots. These modules can be UDP clients, multicast clients, servers, or a custom peer-to-peer UDP networking modules. The custom networking module is given in more detail in a later section. Communication devices can send any type of data supported by the ASL Framework, which includes all sensor information and all actuator information. The framework also supports tele-operation of the robot. A custom serialization module is used in the framework to compress all known data types. This is achieved through special indexing, which shows much better compression than standard .NET serialization. Data packets from the custom serialization module are approximately 10% the size of standard serialized objects from .NET.

The platform contains a list of all associated devices. Every device receives the same actuator information and only devices that accept a known actuator will operate on it. All sensor data is sent through the same event, resulting in the platform having a single input and output event structure. The communication side of the platform also has a single inbound and outbound event for traffic. This has an important consequence—any platform can be swapped with another platform so long as the expected input and output types are the same. Thus, we can run the same algorithm on many different types of robots, or use a simulator with no known difference to the algorithm. For example, the virtual platform used in the screenshots in Figure 69 is a specific platform that reads sensor logs taken at an earlier time and replays the data in the exact same way it was received the first time.

9.2.3. Algorithms

Algorithms are designed just like platforms, but with mirrored inputs and outputs. Algorithms input sensor data and output actuator data. Algorithms have the same communication inbound and outbound paths as platforms. Aside from the event connection scheme, algorithms are quite different from platforms. Algorithms have functional units, which are similar in theory to devices, except functional units all have the same input and output of actuators, data, sensors, and communication. Essentially a functional unit has four inputs and outputs of these specified types. An algorithm can arrange these functional units in any way creating interesting data paths.

An algorithm contains several stages. First at construction, all functional units are created and linked. These links can be changed at anytime during system operation; however, they are typically linked to functions that can switch the receiver of the data instead of switching the link itself. After construction, the platform begins to send data. This data flows into the algorithm and is sent to a sensor handling function and a communication handling function; these functions directs all incoming sensor data and communication data in the algorithm. Once data enters a functional unit, it is passed from one unit to another using the event architecture. Finally, functional units that send actuator or communication data outbound can link to the algorithm itself. The event architecture is vital to the operation of the ASL Framework.

9.2.4. Networking

A network manager module was created for the ASL Framework. This module works with up to 16 systems. It also allows computers to be configured using an IP address and an associated IP address. The network organization of the robot routers is shown in Figure 70. The network manager, which is a communication device in the framework, is configured with the IP address of each computer. For example, we will configure ID 1, which is robot 1, to IP address 10.2.1.100. The diagram shows the IP address for any robot in the system. Laptops and other attached devices will get a DHCP IP address. The network manager allows connections to fail and automatically reconnects when a system rejoins. It also allows the system to function regardless of connectivity. The module also reports network information such as ping time and connection status.

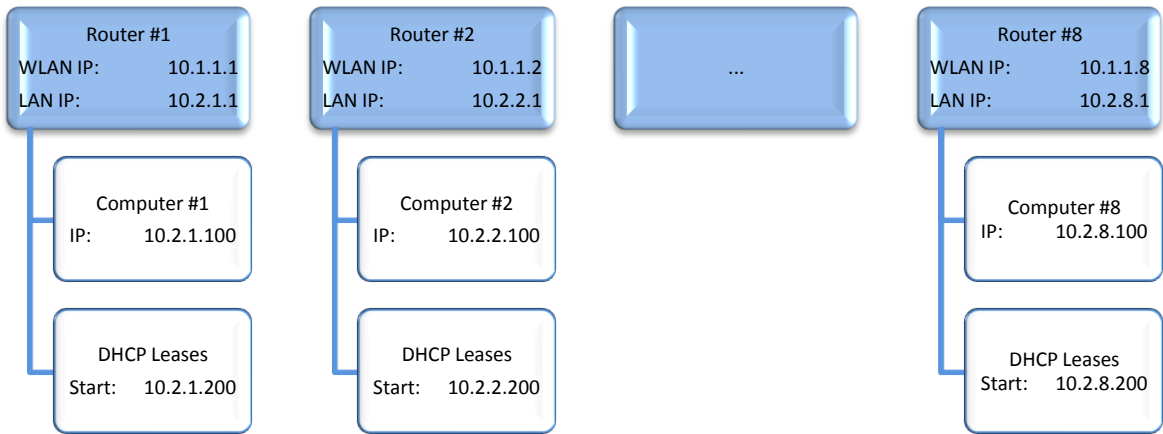


Figure 70 - ASL network organization [87]

9.3. Software Components Developed for Cooperative SLAM

This section contains brief descriptions, screen shots, and a code diagram for all the components that are assembled to execute Cooperative LP-SLAM in the ASL Framework. Figure 71 shows the diagram of connecting devices and functional units in the ASL Framework. The blue boxes indicate devices on the Tamiya platform. The black boxes indicate functional units in the Cooperative LP-SLAM algorithm. The connecting arrows indicate the flow of information over the event architecture.

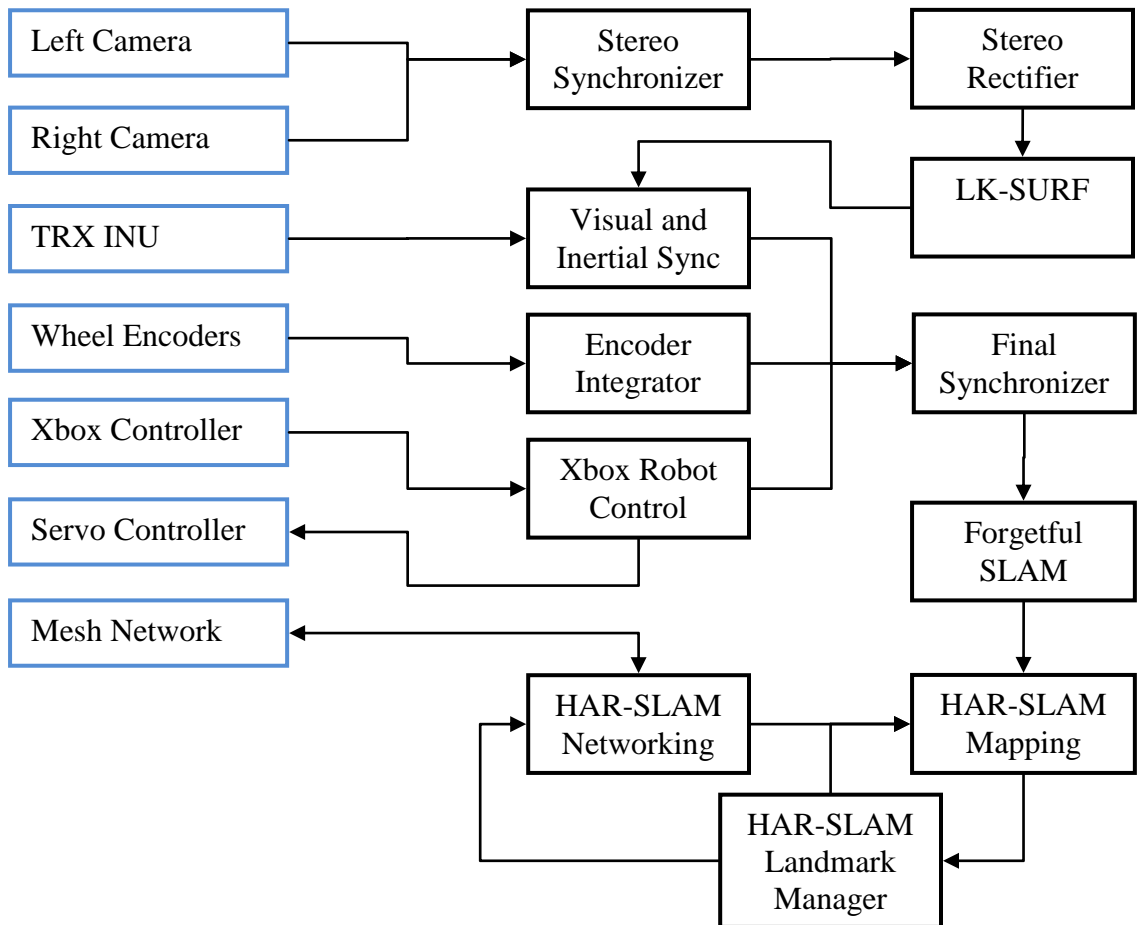


Figure 71 - Diagram of connecting devices (blue) and functional units (black) for Cooperative LP-SLAM

9.3.1. Tamiya Platform

The Tamiya platform is comprised of a set of devices. These devices all communicate in parallel with an algorithm. There are:

- Two Point Grey camera devices for stereo camera acquisition
- A TRX INU device to interface with the INU
- A wheel encoder device to capture wheel encoder data
- A sonar acquisition device to acquire sonar values
- A Parallax servo device to send commands to the parallax servo controller
- An Xbox controller device to interface to an Xbox USB controller
- A mesh networking device to communicate with other robots

The Point Grey camera device uses a third-party library provided by Point Grey Research to interface with the cameras[92]. The device allows for fine tune modification of exposure time, brightness, frame rate, frame resolution, and more. Figure 72 shows a sample screenshot of the device status on the left. On the right of the figure, various camera options provided by the Point Grey Research library are shown. Each image is captured, given an ID specifying which camera the image originated from, and is sent to the algorithms.

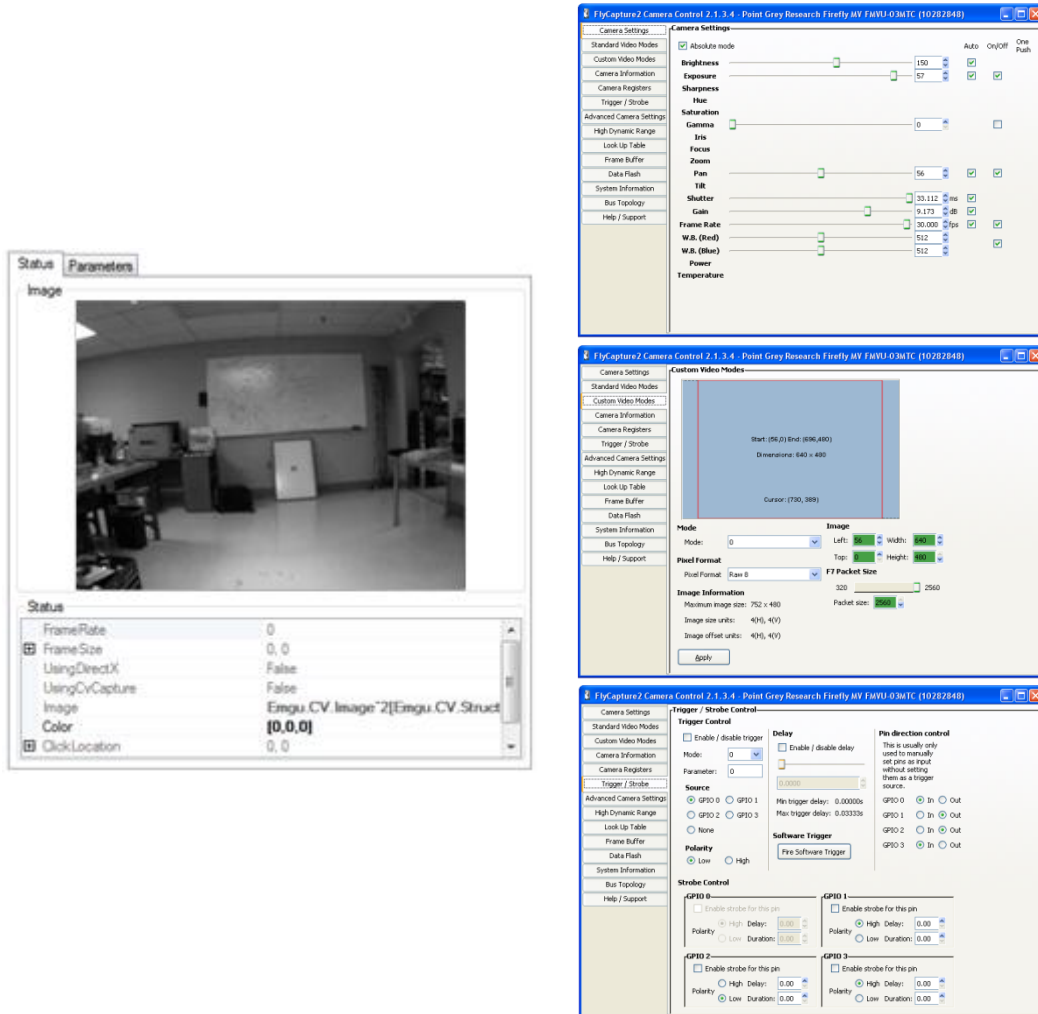


Figure 72 - Point Grey camera status and settings panels

The TRX INU device uses a third-party library provided by TRX Systems. The module connects to the INU over a USB serial port. The only settings required for this device is the COM port number. The TRX INU is manually turned on, and cannot be turned on by the ASL Framework. The TRX INU has been modified to connect to the Point Grey cameras, allowing all three devices to be synchronized by a clock pulse provided by the INU. The software module does not provide an interesting view of inertial data. The device captures raw inertial data, calibrated inertial data, and quaternion orientation data. All the data is sent to the algorithms in sensor packets.

The wheel encoder device connects to the sensor acquisition board developed by ASL. The connection is over a USB serial port. There are two settings for this device, the first is the COM port number, and the second is the wheel encoder calibration. The calibration allows data to be returned in meters instead of arbitrary units. Similar to the TRX INU device, there is no interesting status screen.

A sonar acquisition device is contained in the Tamiya platform. The SLAM algorithm of this dissertation does not make use of this device; however, other algorithms that perform obstacle avoidance use this device. The device connects to the sensor acquisition board developed by ASL. The connection is over the USB serial port, and the only setting is the COM port number.

The Parallax servo device interfaces with the Parallax Servo Controller. There is no third-party library to interface with this device. Similar to the TRX INU device, wheel encoder device, and sonar device, a COM port is used for serial communication. A set of known instructions is provided by Parallax to instruct the device to output pulse-width modulated (PWM) signals. This device is used to control both the steering servos and motor controller. The device does not provide sensor output; instead, it accepts actuator input from an algorithm. Figure 73 shows a sample screenshot of the Parallax servo device status and parameter panels from the ASL Framework. Actuator outputs range between -1 and 1 for servos. The status shifts the position of the markers depending on the actuator instruction. The parameter panel is for settings; settings include COM port number, servo channel assignments, and servo calibration information.

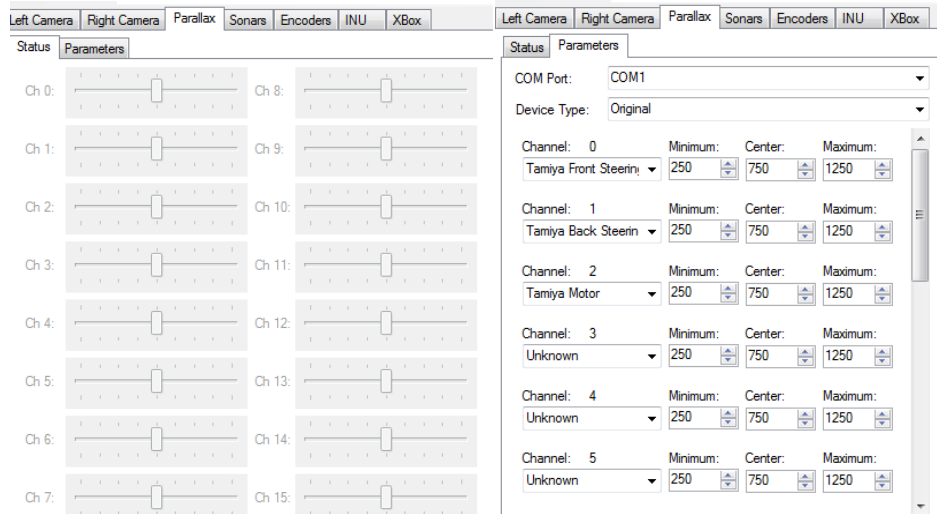


Figure 73 - Parallax servo controller device status and parameter panel

The Xbox controller device uses the Microsoft XNA Studio library to connect to the Xbox controller [93]. This device is used for manual control of the robot. Figure 74 shows the status panel of the Xbox device from the ASL Framework. The parameter panel only contains settings for the controller number to read from and the identifier to attach to sensor packets.

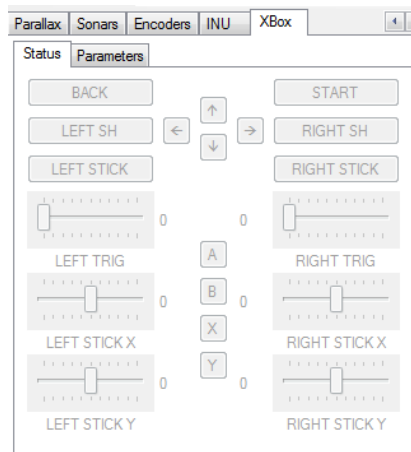


Figure 74 - Xbox controller device status panel

The mesh networking device is the only device used for communication on the robot. It handles incoming and outgoing communication data from the algorithm. The

device uses UDP socket connections between robots (see Section 9.2.4: Networking). Figure 75 shows the status and parameter panel of the device. The status reports if each connection is inactive and the ping time to each robot. The parameter contains settings for the local ID number, the ID numbers of other robots, and the IP address of other robots. The connections are allowed to break at anytime; the only constraint on the design is that IP addresses and IDs are assigned manually.

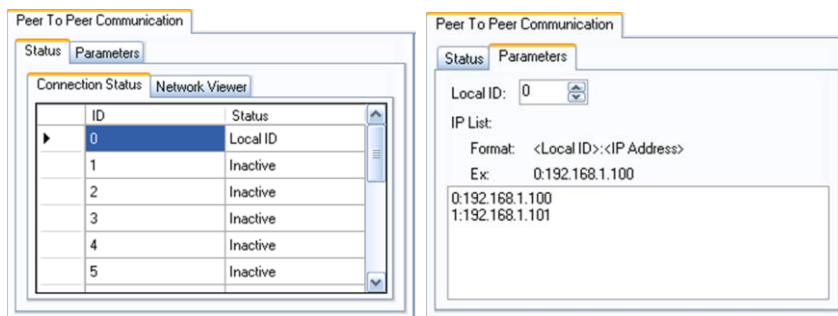


Figure 75 - Mesh networking device status and parameter panels [87]

9.3.2. Landmark Promotion SLAM Algorithm

The LP SLAM algorithm contains a collection of functional units organized as a block diagram. Figure 71 shows the various functional units contained in the LP-SLAM algorithm and the connections between each unit. The following functional units are used:

- Stereo synchronizer, used to create a single stereo image packet from two camera sources
- Stereo rectifier, removes distortions from images in a stereo image packet
- LK-SURF feature tracker, performs LK-SURF routine on incoming stereo images

- Forgetful SLAM, applies the Forgetful SLAM routine to the incoming actuator, inertial, and visual data
- HAR-SLAM mapping, performs HAR-SLAM updates on poses and landmarks
- HAR-SLAM landmark manager, stores landmarks spatially in a hash grid, detects loop-closing, sends updates to HAR-SLAM mapping
- HAR-SLAM networking, communicates with other HAR-SLAM networking functional units, determines global coordinates, sends updates to HAR-SLAM mapping
- Xbox control, converts Xbox sensor data into actuator data
- Encoder integrator, sums wheel encoder distances
- Visual and inertial sync, synchronizes packets from LK-SURF and INU device
- Final synchronizer, synchronizes actuator, encoder, inertial, and visual data into a single packet

The stereo synchronizer functional unit combines separate images from two cameras into a single packet. It matches images based on timestamps. If images are from synchronized cameras, such as the Point Grey cameras, then strict matching can be enabled, giving a small window of 10 ms in order for images to be paired. Regular stereo matching finds the closest timestamp per image. Figure 76 shows a sample snapshot of the stereo synchronizer. In the LP-SLAM algorithm, it combines the raw image from the Point Grey cameras.

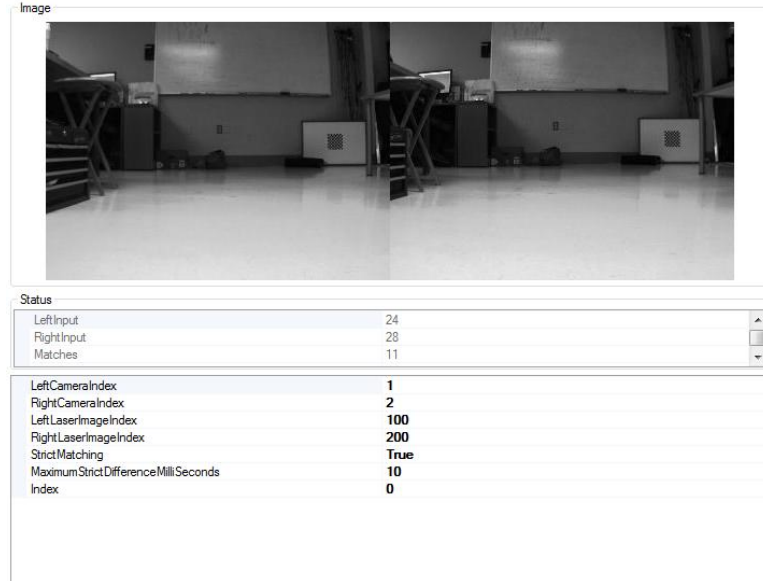


Figure 76 - Snapshot of stereo synchronizer functional unit

The stereo rectifier functional unit removes lens distortion from images. Using the camera calibration parameters (see Section 10.2: Stereo Camera Calibration and Error Modeling), the functional unit applies image rectification routines from OpenCV [22]. The only parameter to the rectifier functional unit is the stereo calibration file.



Figure 77 - Snapshot of stereo rectifier functional unit

The LK-SURF functional unit uses the LK-SURF algorithm described in this dissertation (see Chapter 3: LK-SURF). Each feature is given a unique identifier and a

descriptor from SURF. The feature is tracked over time. For each pair of new stereo images, the known features are tracked, and when features are depleted, the SURF stereo matching routine is called to replenish the list. The list of currently tracked features is sent in a packet to the vision and inertial synchronizer. The only parameter to LK-SURF is the stereo calibration file, which is used to calculate the epipolar geometry. Calculating the epipolar geometry allows the tracker to remove features that do not align with the calibrated images. Figure 78 shows a pair of stereo images with marked locations of SURF features being tracked on both images.

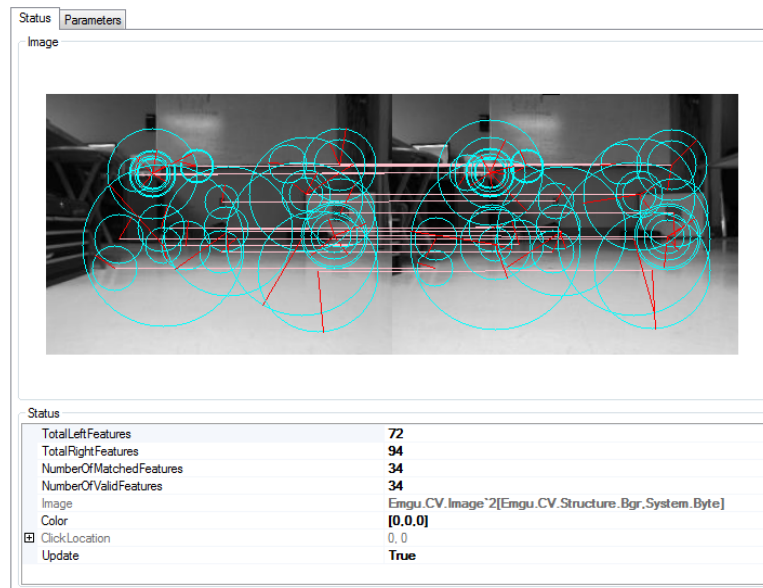


Figure 78 - Snapshot of LK-SURF functional unit

The Forgetful SLAM functional unit accepts packets of synchronized data that contains actuator information, encoder distance, inertial orientation, and the list of currently tracked features. The functional unit performs the Forgetful SLAM routine as described earlier in this dissertation (see Chapter 6: Forgetful SLAM). This functional unit uses a third-party library from Mathworks called the Matlab Common Runtime

(MCR)[94]. This allows the functional unit to call SLAM routines written in Matlab. This functional unit does not have an interesting display as data is passed to Matlab.

The HAR-SLAM mapping, landmark manager, and networking functional unit also uses the MCR library. Since data from Forgetful SLAM is stored using the MCR library, and HAR-SLAM routines were developed in Matlab, it is easy to just call the routines from the functional unit rather than implement them again.

The HAR-SLAM mapping functional unit takes in new landmarks and poses from Forgetful SLAM. It also receives updated landmarks from the HAR-SLAM landmark manager, and receives landmark updates from the HAR-SLAM networking functional unit. In this functional unit, all data is organized using graph structures. Updates of landmarks are sent to the HAR-SLAM landmark manager and the HAR-SLAM networking functional unit if they pass certain criteria (see Chapter 8: Cooperative SLAM with Landmark Promotion).

The HAR-SLAM landmark manager functional unit organized landmarks spatially using a hash grid. All landmarks come from the HAR-SLAM mapping functional unit. The role of this functional unit is to close-the-loop by comparing incoming sets of landmarks to its spatial database. When a match is found, the pair of matching landmarks is sent back to the HAR-SLAM mapping functional unit.

The HAR-SLAM networking functional unit receives updated landmarks from HAR-SLAM mapping. It is the only functional unit to use the communication events. It sends landmarks to the other robots. The landmarks of each robot are compared to every other robot. When a match is determined, the match is shared amongst all robots. Each

robot locally performs the global coordinate update filter as it deterministic (see Section 7.2.3 Multiple Robot Updates). The global coordinate update results in landmarks being updated (see Section 8.1.3: Map merging). The updated landmarks are sent to HAR-SLAM mapping.

The Xbox robot control functional unit translates Xbox controller input information into actuator data for the Tamiya. This functional unit has no parameters and does not have an interesting status panel.

The encoder integrator functional unit accepts wheel encoder packets. The functional unit adds up all wheel encoder data and sends out an integrated wheel encoder packet instead of the normal packet. This is done to prevent data loss from accidental packet drops. The ASL Framework is event driven, and if a functional unit has a backup of incoming packets, the packets will be dropped to prevent system lag. If a packet is dropped, having the total distance travelled allows interpolation to estimate distance travelled.

There are two more synchronizer functional units. The vision and inertial synchronizer matches feature packets from LK-SURF to inertial data from the TRX INU. A matching routine similar to the stereo synchronizer is used to pair packets. The paired packets are sent to the last synchronizer. The last synchronizer combines actuator data from the Xbox controller, integrated encoder data from the encoder integrator, and the pair of inertial and visual data from the previous synchronizer. The wheel encoder data is upsampled to match the frequency of the visual and inertial data packet. The INU, Xbox controller, and cameras operate at 20 Hz, but the wheel encoders operate at 15 Hz. Using

the integrated wheel encoder results, the encoder information is upsampled using linear interpolation to match the 20 Hz signal. The final synchronizer sends a single packet, containing all the incoming data types, to the Forgetful SLAM functional unit.

CHAPTER 10: CALIBRATION AND MODELING

Calibration and modeling are vital when implementing algorithms on actual platforms. This chapter covers the techniques used for calibrating and modeling the test platform described in the previous chapter. In addition to calibration, modeling sensor noise is covered. The sensors that are calibrated and modeled include the wheel encoders, stereo cameras, and inertial measurement units. Special calibration considerations are taken for the Optical INU. Finally, the robot platform itself is modeled.

10.1. Encoder Calibration and Error Modeling

Encoders have only one primary role in the Tamiya-based platform, and this is to measure distance. Since the wheels are not rigidly aligned to the robot platform, encoders provide a poor measure of rotation. Rotation measurements are left to the inertial measurements. The encoders simply return a count of the total of lines that pass on the wheel strip in a particular interval. The encoder has the ability to discriminate between backward and forward motion; the total count is either added or subtracted depending on the motion. Ideally, we could just measure the diameter of the wheel, calculate the circumference, and divide by the total number of counts per rotation; however, in practice this is not accurate.

The encoders are easy to calibrate, we simply have the robot follow a straight line for a known distance, and count the total number of encoder lines measured. A longer distance will produce a more accurate calibration. This process can be repeated several times in order to produce an accurate result. Figure 79 shows the encoder calibration path used for the robot. A simple red line following algorithm was used with a PID controller to keep the robot as precisely on the line as possible. The length of the path was 18 meters, and the calibration was repeated several times for accuracy. The encoder calibration for one robot was 0.4655 mm per encoder value on the front left tire, 0.4655 mm per encoder value on the front right tire, 0.4638 mm per encoder value on the back left tire, and 0.4666 mm per encoder value on the back right tire.



Figure 79 - Encoder calibration environment [83]

For error modeling, we need to test different scenarios and determine an approximate model to apply that best describes the noise. Several logs need to be taken at constant speed such that there is a single target mean. In the case of the encoders, we seem to have a consistent result. Most of our samples successfully fit Gaussian models, with the standard deviation linearly related to the mean at a ratio of approximately 0.02.

The Kolmogorov–Smirnov Test (KS-Test) was used to validate the approximations [78]. Figure 80 shows an encoder dataset plotted in a histogram with a Gaussian model fitted.

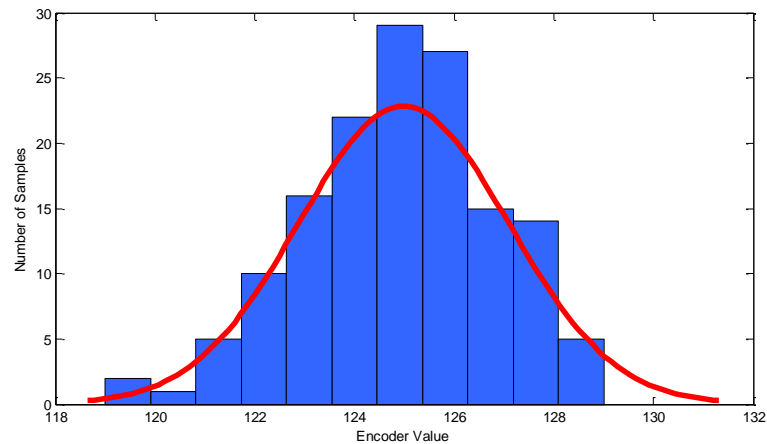


Figure 80 - Histogram and Gaussian fit for sample encoder calibration

10.2. Stereo Camera Calibration and Error Modeling

A stereo camera is much more complicated than an encoder is. While an encoder only measures one dimension, a stereo camera system measures three. In order to use stereo cameras, we need to calibrate each camera, and calibrate the stereo pair. Errors in stereo camera readings propagate from errors in each separate camera image, and from false matches. Only errors in the image can be analyzed, false matches are completely non-Gaussian and have no means of prediction. The Robust Kalman Filter takes care of the false matches.

10.2.1. Calculating Camera Calibration

A calibration routine is required for any accurate optical systems. Two libraries were tested. The first is a toolbox based in Matlab, and the second is the Intel OpenCV

library. The latter was selected due to its accuracy and speed. The Matlab toolbox has nice visualizations, while OpenCV has none. Figure 81 shows a sample calibration result from the Matlab toolbox. It is interesting to see the various checkerboard positions in relation to the camera (left) or the camera positions in relation to the checkerboard (right). In order to use the Matlab toolbox, each checkerboard had to be manually added to the toolbox, and each image required the user to select the four primary corners in the same order on every image.

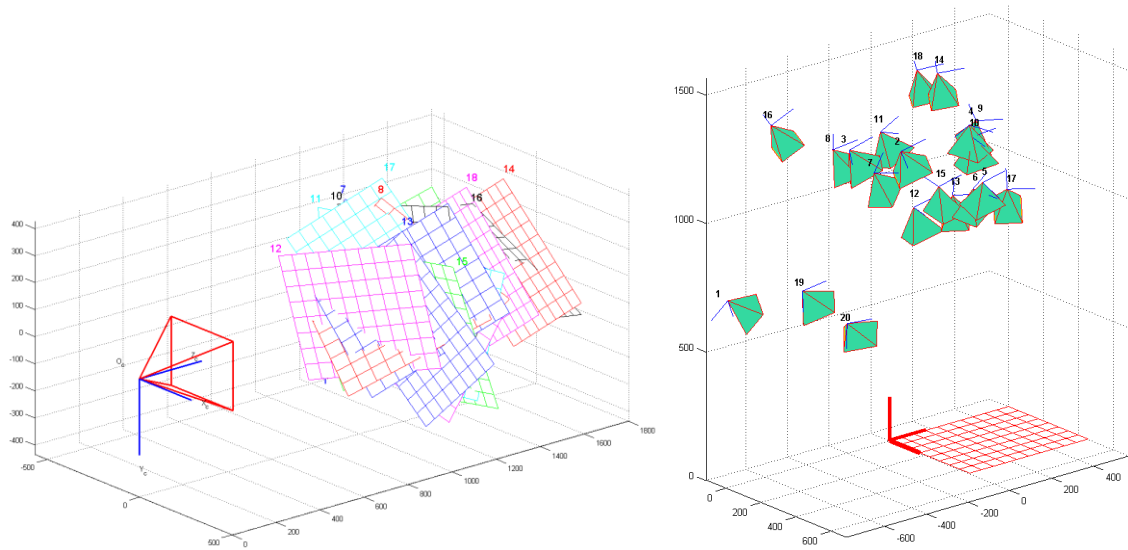


Figure 81 - Matlab toolbox calibration - camera-centered (left) and world-centered (right) views given multiple checkerboard samples

The OpenCV library has built in checkerboard finding routines. With some extra programming, a visualization was created, as is shown in Figure 82. The software automatically determines the four primary corners, and all the corners in between. Instead of using a small set of calibration images taken manually, the ASL Framework version of stereo camera calibration—using OpenCV functions—is able to continuously

sample a live video, extract checkerboard data, and finally calculate all of the calibration parameters on hundreds of images.

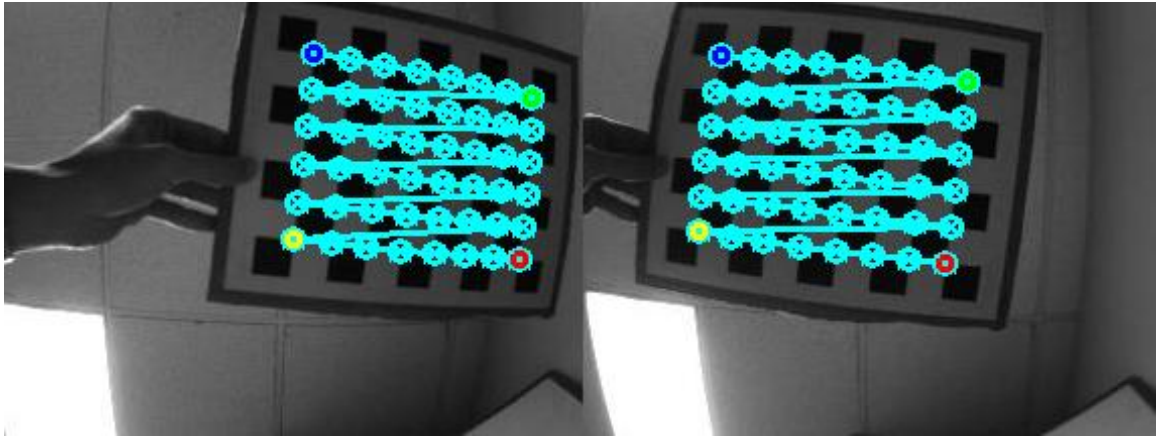


Figure 82 - Screenshot of stereo camera calibration in ASL

The ASL Framework stereo camera calibration routine performs individual camera calibration, and then uses the same data again to perform stereo camera calibration. This stereo calibration information is used to rectify the captured checkerboard data and perform additional stereo calibration on the rectified data. This results in two sets of calibration data. The second set of calibration data is used to triangulate features after the images have been rectified. The next section explains rectification.

10.2.2. Rectifying Stereo Images

Rectifying images involves removing lens distortions as calculated by the camera calibration, and aligning the images as if it were a perfect stereo pair. A perfect stereo pair would be exactly aligned in orientation and on the same horizontal axis, thus any common point in the pair of images only shifts sideways between the images. Having

this property can reduce computation time by a significant amount since the images are already aligned. The stereo image pair in Figure 83 shows the raw captured images from the stereo cameras on top. Notice the distortions in the ceiling tiles, and the location of objects in different vertical positions.

Using a set functional unit in the ASL Framework that applies the OpenCV rectification, the same raw images are combined with the calibration information. This yields the image shown on the bottom in Figure 83. The rectified image has the distortions removed, the cameras appear to be oriented in the exact same manner, and all objects are straight and aligned vertically.

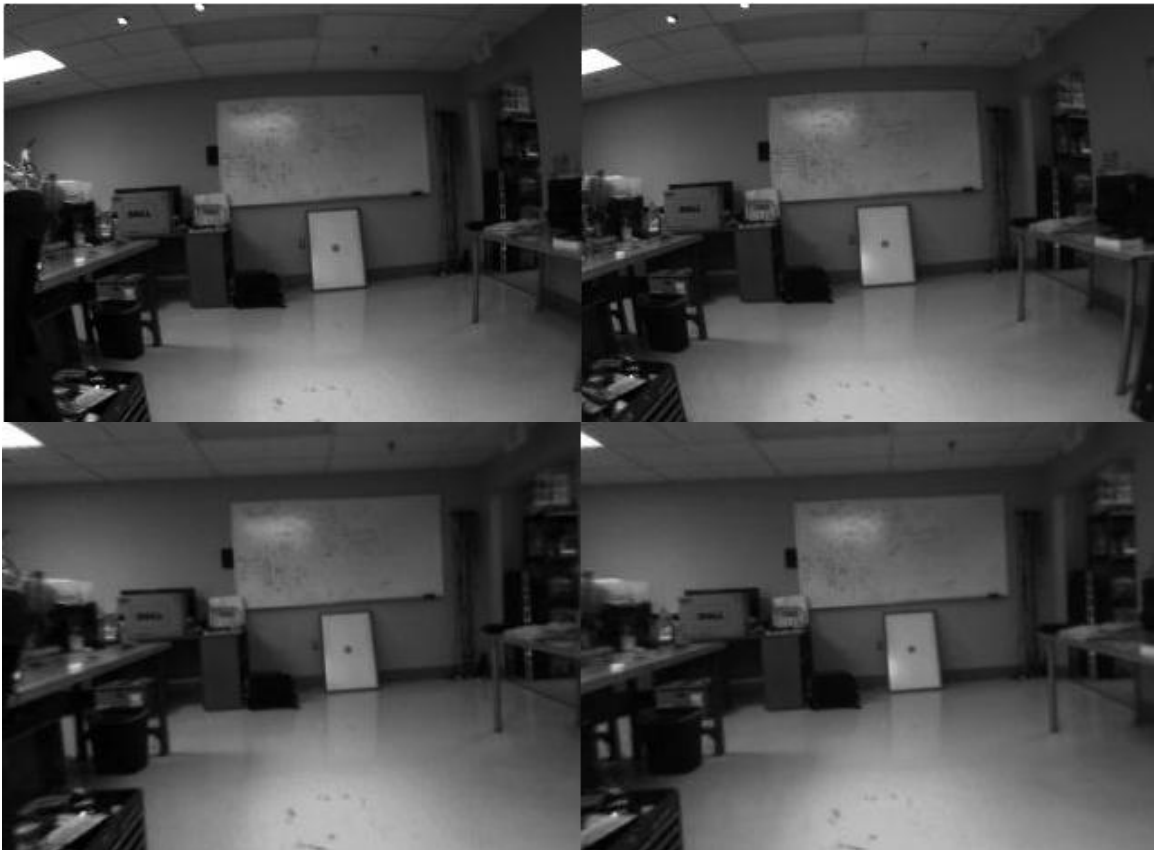


Figure 83 - Top: Sample unprocessed stereo image pair; Bottom: Rectified stereo image pair

The camera calibration routine provides the information necessary to perform the rectification. However, once rectified, the cameras behave as if they have a new set of calibration parameters since they have been moved virtually. Though the data source has not changed, the images are transformed resulting in new calibrations being required. The camera calibration routine considers this, warps the checkerboard information according to the first calibration, and calculates a secondary set of calibration information for all functions and algorithms that make use of the rectified images instead of the raw images.

10.2.3. Modeling Error

When performing stereo triangulation, there are two sources of error. The first is the precision and accuracy of the selected points. In a physical system, sensor information is prone to errors. Estimating the accuracy of finding a feature in an image is one important source of error. The other source of error is falsely matching features. This error cannot be predicted since it is based on higher-level routines that use Boolean logic versus mathematical equations.

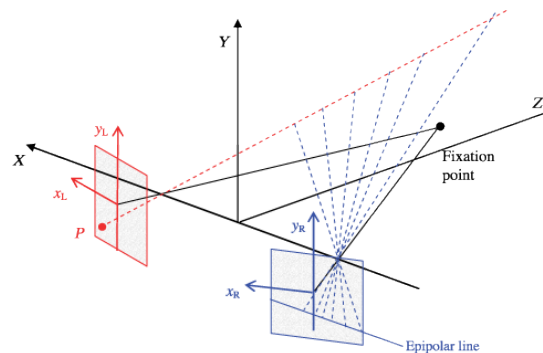


Figure 84 - Diagram of an epipolar line [15]

In order to model the error of a stereo camera, the noise in selecting a feature is analyzed. Using epipolar geometry an estimate of the true location of a feature can be estimated using its stereo match. Figure 84 shows how a point in the red image projects a ray into the real world. On the blue image, this ray is seen as a line, which indicates all the possible locations for a feature in the blue image. This information can be used to estimate a feature's distance from the ideal epipolar line given the location of its stereo pair.

In order to determine what feature should match without any chance of error, the checkerboard finding routine is revisited. It is quite simple to find a checkerboard in an image given the OpenCV routines; matching two checkerboards to each other for stereo correspondence is also easy, assuming that the checkerboard is not square, and the cameras have relatively similar orientations. A collection of matched points is logged; these points are all selected in a similar manner to how other features in this dissertation are selected. Each pair of features gives us a point in the left and right image, and by using the calibration of the cameras, the epipolar line is determined. The distance each point is from the line is shown in Figure 85. A Gaussian Mixture Model of two Gaussians is used to describe the distribution accurately, and passes the KS-Test.

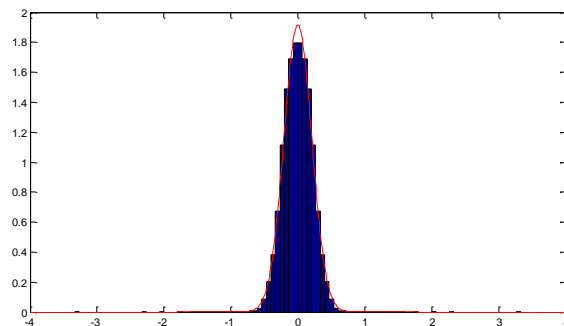


Figure 85 - Histogram of epipolar-based errors with a Gaussian Mixture Model fit

Having an error model fit the pixel error shown in Figure 85 does not solve the problem of modeling the triangulation error. It is not practical to capture every possible location in front of a stereo camera set to produce an accurate error distribution; instead, the data is synthesized. By selecting a random point in one camera's image, the expected epipolar line can be determined in the other image from the calibration. Now, randomly select a point on that line, then given these two random points, a projection can be made in three dimensions. Given a point on each image there is now a valid projection representing the "true location." Randomly select points away from each of the two points using the pixel error model, then find the 3D projection. The error is the difference between this projection and the "true location" projection previously computed before adding the pixel error. This technique can be approximated using the Unscented Transform, a linearization of the projection, or using many samples.

In the case of this dissertation, a Gaussian Mixture Model was propagated using the Unscented Transform. Figure 86 shows only the distance projection from the cameras. The horizontal and vertical angular measurements from the camera matched similar error models as the camera pixel error, which is expected based on the transformation. The distance calculation from the camera involved several trigonometric functions; this magnifies error as a target is further away.

Figure 86 shows all the random samples scattered over distance away from the camera. The left plot shows the mean of the Gaussian error model at each random sample. The graph is not clearly a single line as the position of the target from left to right plays a role in accuracy, however a best-fit power line approximated the model mean at a given distance. The right plot shows the variance of the error model given

distance; after 10 to 15 meters, the variance becomes too great to be of any use. Physically this makes sense as the camera's low resolution and small separation between the stereo camera pair prevents it from accurately ranging at large distances.

The result of the error model is a set of independent Gaussians, with mean and variance given by the curves in Figure 86, which represent the stereo camera ranging error distribution as a function of distance from the camera. Only the Gaussian distributions dealing with distance measure actually varies with distance, the vertical and horizontal angular measurements are independent of distance. The prediction stage of CUMRF was used to propagate the Gaussian Mixture Model of the pixel error and finally consolidate to a single Gaussian distribution.

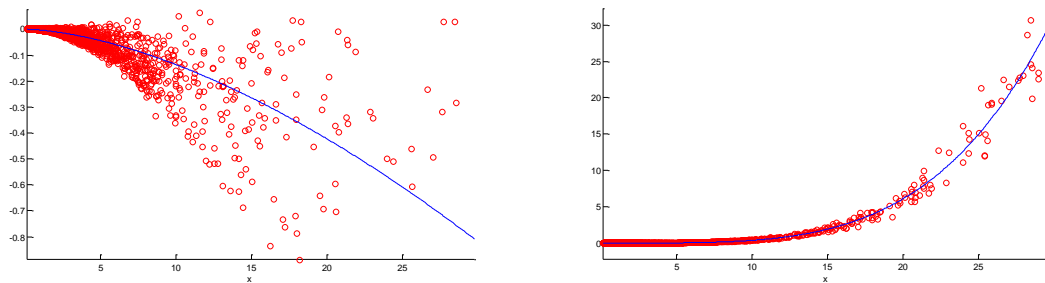


Figure 86 - Stereo camera projection distance mean errors and variances. Left: Scatter of mean error vs. distance. Right: Scatter of variance vs. distance.

10.3. Inertial Error Modeling

The Inertial Navigation Unit (INU) from TRX Systems provides extra an extra level of stability and performance above traditional MEMS INUs. Internally an orientation filter operates to reduce drift and improve the orientation estimate of the INU. Figure 87 shows a basic outline of the orientation filter. The exact mechanics of this filter are not publicly available, but regardless, the device provides an estimate of the

orientation instead of raw angular rate measurements like most inertial measurement units provide.

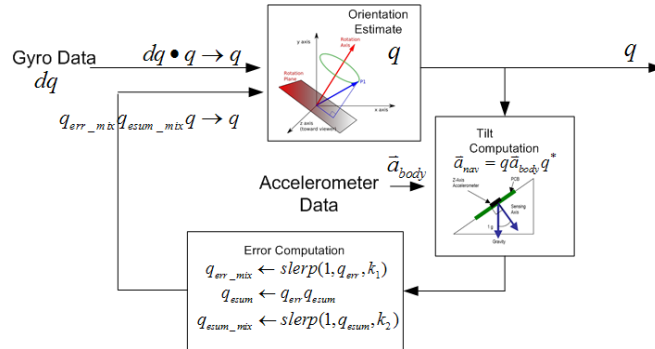


Figure 87 - PI implementation of orientation estimator [9][15][85][84]

Regardless if the unit is stationary or in motion, the TRX INU seems to have a consistent level of random noise that is nearly Gaussian. The INU noise is computed by reading orientation output of the unit over a period of 10 minutes while stationary. The noise is taken as the difference in orientations over a single sample. The system operates at 40 Hz. Figure 88 shows on the left a sample histogram of noise generated by the INU when stationary with a Gaussian fit. It appears to be Gaussian, but fails the Kolmogorov–Smirnov Test (KS-Test) [78]. The right side shows a Gaussian Mixture Model fit, using three Gaussians. Visually the Gaussian Mixture Model seems to pass the test; furthermore, this fit passes the KS-Test, making it mathematically valid. If we wish to be accurate, we use the Gaussian Mixture Model, if we wish to have faster results; we can use the Gaussian approximation.

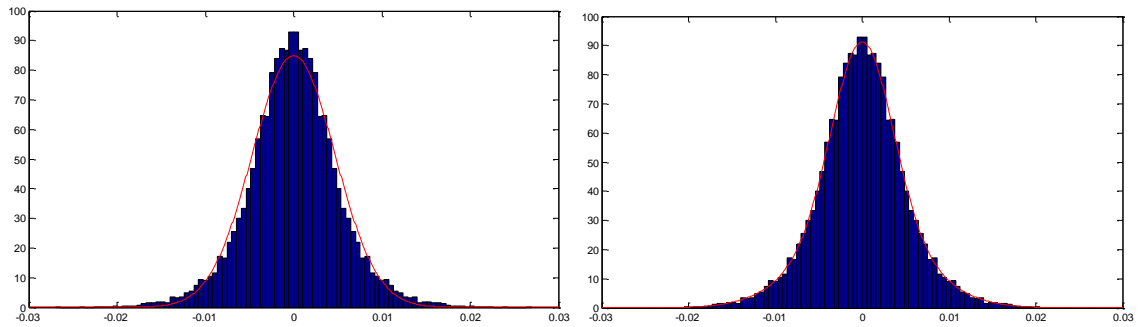


Figure 88 - Histogram of stationary orientation data in one dimension.
 Left: Gaussian Model fit, failed KS-Test, Right: Gaussian Mixture Model fit, passed KS-Test

10.4. Aligning Optical and Inertial Systems

Once both the stereo cameras and the inertial orientation are well calibrated, another step is required before the two systems can be used together. No matter how precisely two devices are mounted together, it is always a good idea to make sure the separate devices have the same coordinate system. Minimally with an orientation device, we must calculate the rotation between the optical and inertial coordinate systems. A simple experiment can be setup, where a rectangular checkerboard is fixed in a position such that the vertical direction of the checkerboard matches the direction of gravity. Normally an optical system has no indication of the direction of gravity, but by aligning this checkerboard, we can take a sample video of the checkerboard and from each frame, we can calculate the direction of gravity based on the direction of the checkerboard. Figure 89 shows a sample view of the aligned checkerboard, notice how the longer side of the checkerboard aligns with gravity.

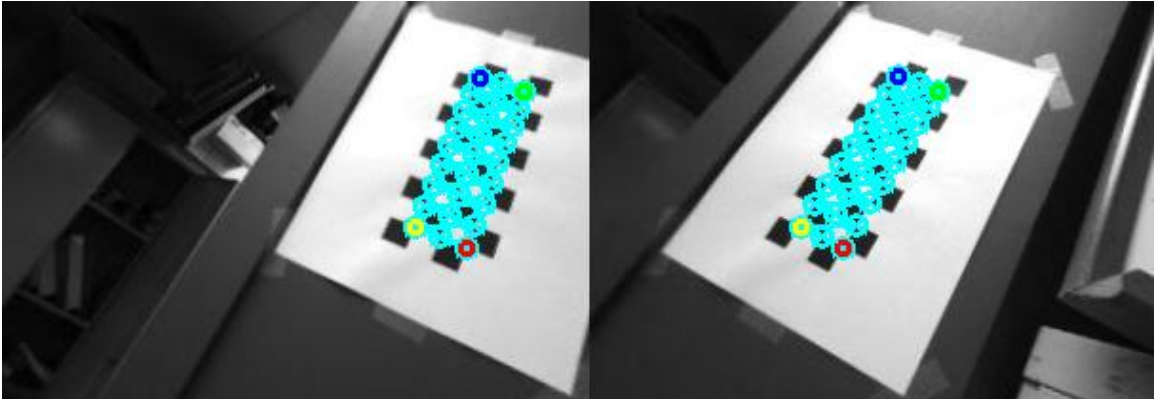


Figure 89 - Stereo view of gravity aligned checkerboard

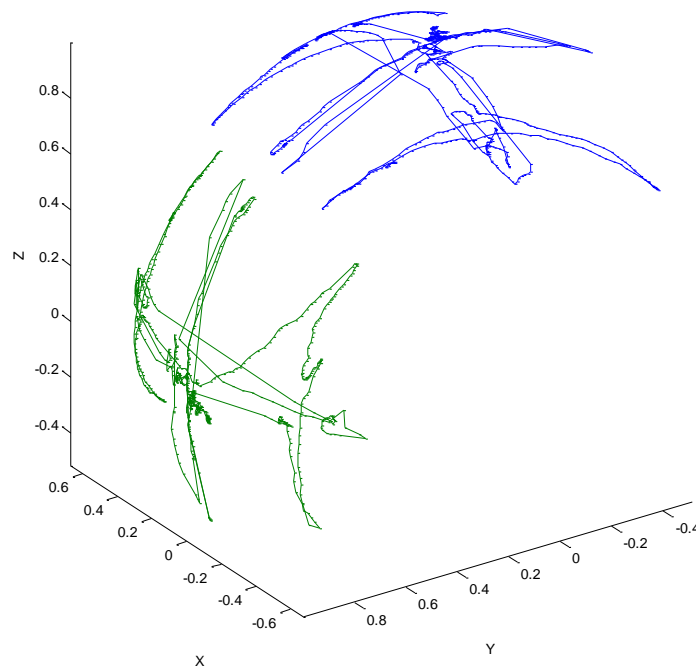


Figure 90 - Inertial (blue) vs. optical (green) directions of gravity on unit sphere

Once a log has been captured, the inertial estimate of the direction of gravity can be compared to the optical direction of gravity. Figure 90 shows the optical (green) and inertial (blue) direction of gravity over the entire log taken. The plot is on a unit sphere, and the paths are the continuous sampling of the direction of gravity over time. There are significant coordinate system differences. These major differences are fixed with a simple rotation. By collecting the entire set of vectors in three dimensions from both

systems, we can use the Kabsch algorithm, which uses singular value decomposition to determine the optimal rotation matrix that minimizes the root mean square deviation[95].

Kabsch Algorithm

Input:(**A**,**B**) Output:(**R**)

A is a matrix of n by 3, and **B** is a matrix of n by 3, where we have a list of 3D vectors

$$\mathbf{X} = \mathbf{A}^T \mathbf{B}$$

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = SVD(\mathbf{X}) \text{ resulting in } \mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T$$

Table 44 - Kabsch algorithm [95]

After running the Kabsch algorithm on this data set, we obtain a rotation matrix that results in the Euler rotations of X = -90.0401 degrees, Y = 0.4555 degrees, and Z = -179.8732 degrees. Notice that these rotations are more than just a change of coordinates, but show the slight imperfections in mounting of less than half a degree. Figure 91 shows the same inertial plot of gravity (blue) with the newly transformed optical plot of gravity (green). The different views of the plot show the paths on the unit sphere to be nearly identical, which is exactly the result we want. Now we simply apply this rotation matrix to all data from the Optical INU in order for it to be in the same coordinate system and the inertial unit.

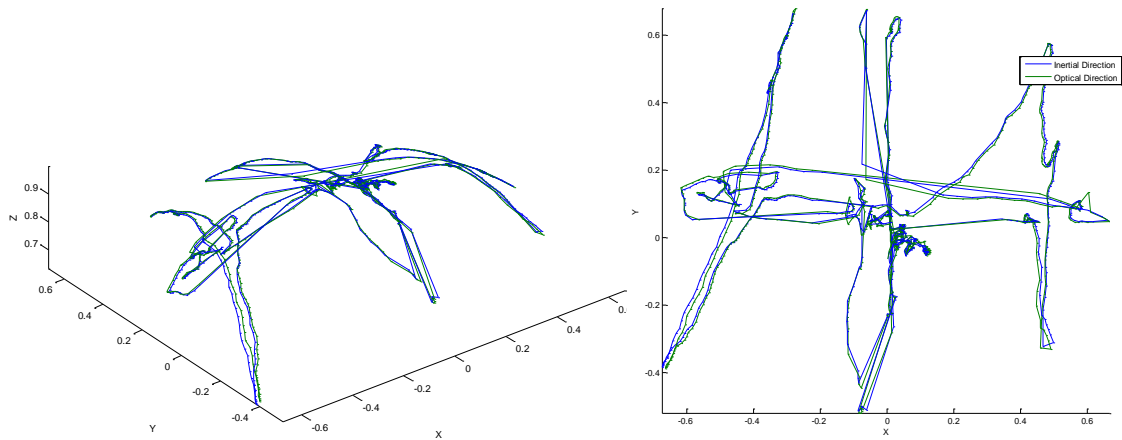


Figure 91 - Multiple views of calibrated inertial (blue) and optical (green) directions of gravity on unit sphere

10.5. Robot System Modeling

The last piece of useful data given a robotic platform is the model of the robot itself. This can be thought of as using the robot control data, the mechanics of the robot, and the state or history of the robot as a new piece of information to improve the filtering of the other sensors on the robot.

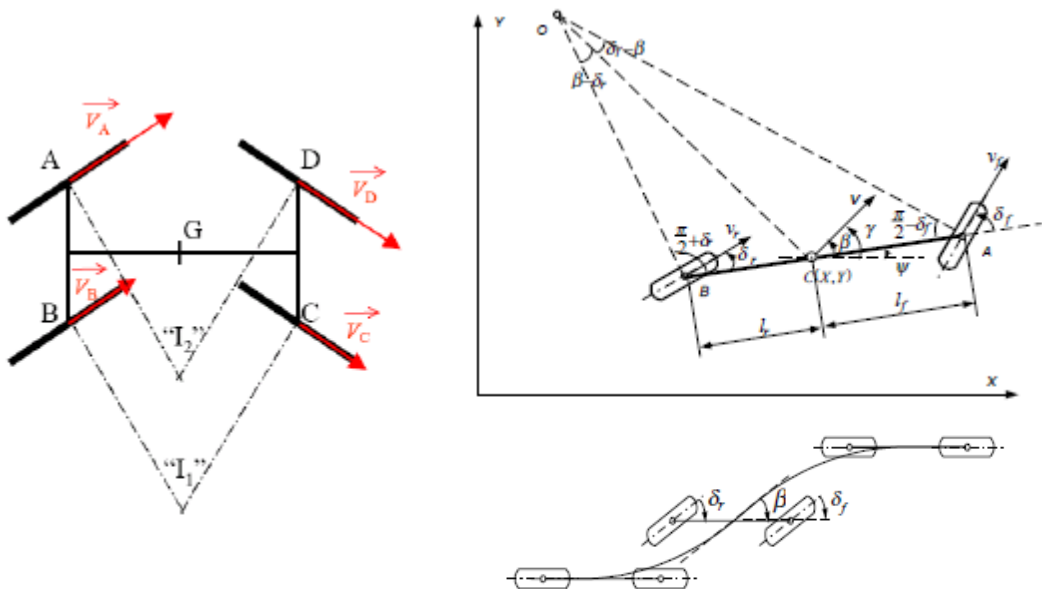


Figure 92 - Left: Four-wheeled vehicle with parallel steering [96]; Top-Right: Kinematic model [97]; Bottom-Right: Parallel steering maneuver [97]

There are many methods to create a model for a robot. The Tamiya-based platform has both front and rear steering, deformable tires, and a suspension system. Sample models shown in Figure 92 are simplified versions of the robot that could be used; however, there are many parameters that require estimation or measurement. Complex models require parameters such as, the center of mass, spring coefficients, mechanics of the suspension, tire parameters, friction coefficients, and more. Calculating the parameters of these models requires either careful measurements or gradient descent techniques with near perfect data.

An alternate method to producing a model is to use a neural net. A neural net works well as a function estimator, and given both input and output data a model can be built. In the case of modeling the robot in this dissertation, the control input from the ASL Framework and a small history of sensor input is used as a control input and state input. The expected output can be the next sensor input, or an external source indicating the robot velocity and change in angle.

For simplicity, the TRX INU is used to give both expected output and historic input. The collection of calibrated encoders is used to give the expected output as a mean of all the encoders. The input is a history of each individual encoder and a history of the change in quaternion estimate from the INU. Finally, a history of the control input is given. A neural net attempts to estimate the output as best as possible given the input, but does not match every output exactly. The noise introduced from using sensors as output is negligible with a lot of training data [98].

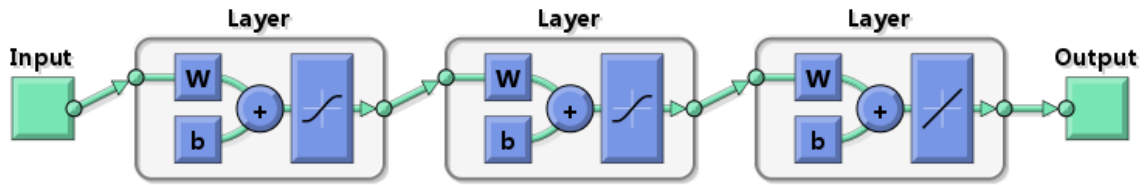


Figure 93 - Diagram of three-layered neural net to describe robot model

Many neural net structures exist; however, feed-forward networks seem to work best as function estimators. A single layered neural net with a linear transformation step is the exact same thing as a linear system with a single matrix and a vector offset. This is used to train and calculate a linear model to compare to models that are more complex. Tangent sigmoids are useful transformation functions for estimating nonlinear functions. In order to estimate convex functions, two layers are necessary; the first layer has a tangent sigmoid transformation and the second layer has a linear transformation [98]. Most functions can be estimated with two layers, but non-convex functions, which are more complicated, require three layers. The first two layers both use tangent sigmoids, while the last layer uses a linear function [98]. Figure 93 is a diagram of a three-layered neural net. The number of neurons per layer can arbitrarily be chosen; however, overestimating will usually work out better. A brute force technique was used to train the system model. The method selected randomly the number of layers, the number of neurons per layer, and the number of historic samples needed as input. The history range was chosen between one (the last value) and five (the last five values). Figure 94 shows a screenshot of the Matlab toolbox used to train neural nets. The brute force method exhaustively trains many different neural nets and keeps the best performing one.

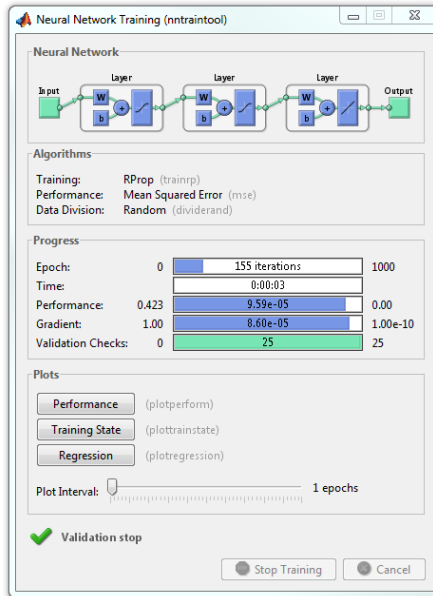


Figure 94 - Screenshot of Matlab Neural Network Toolbox

10.5.1. Comparing linear vs. nonlinear models

The results of the proposed training method produced a useful neural net with three layers and a history of only the previous sensor values. Over 15,000 training samples were used to create the models. The same method was used to force a single layered linear system. The linear system only uses the previous sensor values. Figure 95 and Figure 96 show sample paths where the black line is the estimate of the true path taken, the blue line is generated from the linear model, and the red line is generated from the nonlinear model. The paths generated by the models are prediction-only paths; no corrections are made.

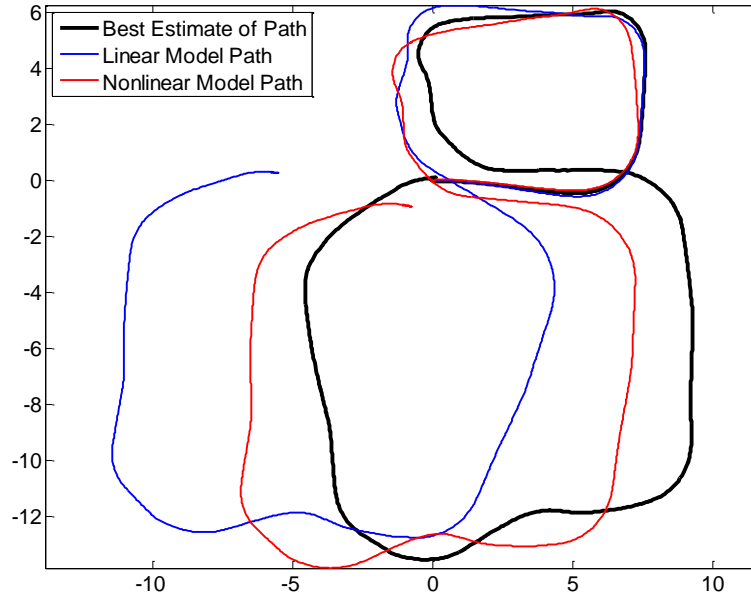


Figure 95 - Test path of figure-eight around lab, units in meters. Black path is the best estimate of the actual path taken. Blue path is generated only by linear model predictions. Red path is generated only by nonlinear model predictions.

The test path shown in Figure 95 is a figure-eight style path, first a small counterclockwise loop is made followed by a larger clockwise loop. In Figure 95, the linear model produced a path with 10.4% error over distance travelled. The nonlinear model produced a path with 2.5% error over distance travelled. The test path shown in Figure 96 is a set of four loops, clockwise around the lab. In Figure 96, the linear model produced a path with 7.1% error over distance travelled and the nonlinear model produced a path with 3% error over distance travelled.

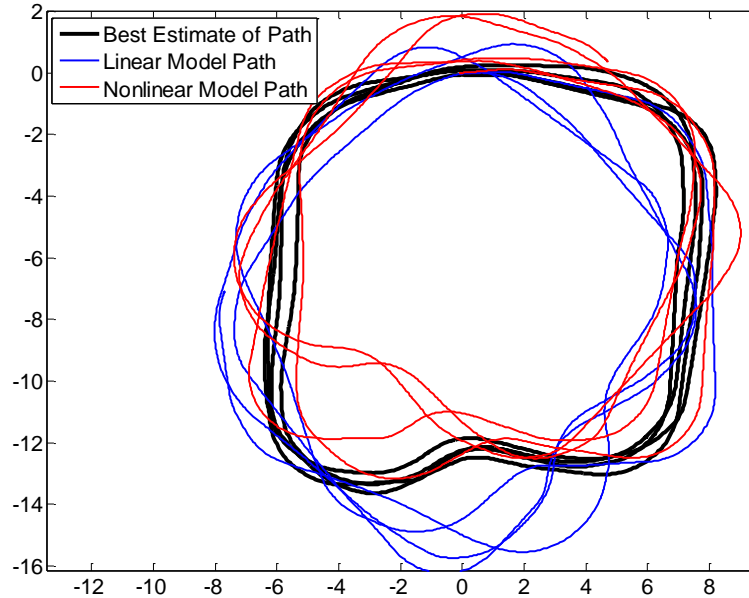


Figure 96 - Test path of four loops, units in meters. Black path is the best estimate of the actual path taken. Blue path is generated from linear model predictions. Red path is generated from nonlinear model predictions.

10.6. Robot State and Observation Models

Once dynamics and error models have been created, the final state model and observation equations can be developed. The state contains elements pertaining to the pose of the robot, the landmarks contained in the map, and the previous robot pose. The previous pose is included for two reasons. The first relates to predicting observations, the second related to HAR-SLAM.

The previous section covered a neural net model for robot dynamics, which produced output of a velocity-like estimate of the robot given a set of input controls. The output is the distance travelled during a single time step and the rotation of the robot during a single time step. Since the objective of this dissertation is mapping, an appropriate state for the robot (ignoring landmarks) is a 3D location and quaternion heading. Given that the robot used in this dissertation is confined to a planar surface, a

2D location will be used. The quaternion will remain in the state as observed features are in 3D and the robot's mechanical suspension can alter orientation in all three dimensions.

$$\mathbf{x}_{pose} = \begin{bmatrix} \text{location X} \\ \text{location Y} \\ \text{orientation W} \\ \text{orientation X} \\ \text{orientation Y} \\ \text{orientation Z} \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \quad \mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{pose}^k \\ \mathbf{x}_{pose}^{k-1} \\ \mathbf{x}_{landmark}^1 \\ \mathbf{x}_{landmark}^2 \\ \vdots \end{bmatrix}$$

$$\mathbf{x}_{landmark} = [p_x \quad p_y \quad p_z]^T$$

Table 45 - Robot and map state

$$q = [q_w \quad q_x \quad q_y \quad q_z]$$

$$\|q\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

$$q^{-1} = \frac{1}{\|q\|^2} [q_w \quad -q_x \quad -q_y \quad -q_z]$$

$$quatMultiply(a, b) = \begin{bmatrix} a_w b_w - a_x b_x - a_y b_y - a_z b_z \\ a_w b_x + a_x b_w + a_y b_z - a_z b_y \\ a_w b_y + a_x b_w - a_y b_z + a_z b_x \\ a_w b_z + a_x b_y - a_y b_x + a_z b_w \end{bmatrix}^T$$

$$rotM(q) = \frac{1}{\|q\|^2} \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2q_w q_z - 2q_x q_y & 2q_w q_y + 2q_x q_z \\ 2q_w q_z + 2q_x q_y & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2q_y q_z - 2q_w q_x \\ 2q_x q_z - 2q_w q_y & 2q_w q_x + 2q_y q_z & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

$$zrot(q) = \tan^{-1} \left(\frac{2q_w q_z - 2q_x q_y}{q_w^2 + q_x^2 - q_y^2 - q_z^2} \right)$$

Table 46 - Quaternion functions

The pose vector, landmark vector, and complete state vector is shown in Table 45. Table 46 provides common quaternion functions, which are used in state and observation prediction functions. Using a velocity model for the robot[27], the neural net model is

treated as the control input. Table 47 contains the dynamics of the robot state and Table 48 contains the complete dynamics for the robot state, past state, and map.

$$\begin{aligned}
 \mathbf{u} &= \begin{bmatrix} d \\ \omega_w \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \text{neuralNetModel}(\text{rawInput}) \\
 \tilde{\mathbf{q}} &:= \text{quatMultiply}\left(\begin{bmatrix} q_w^{k-1} & q_x^{k-1} & q_y^{k-1} & q_z^{k-1} \end{bmatrix}, \omega\right) \\
 \mathbf{x}_k &= f\left(\mathbf{x}_{k-1}, \begin{bmatrix} d \\ \omega \end{bmatrix}\right) = \begin{bmatrix} p_x^{k-1} + d \cos(\text{zrot}(\tilde{\mathbf{q}})) \\ p_x^{k-1} + d \sin(\text{zrot}(\tilde{\mathbf{q}})) \\ \tilde{q}_w \\ \tilde{q}_x \\ \tilde{q}_y \\ \tilde{q}_z \end{bmatrix} \\
 \frac{\partial f}{\partial \mathbf{x}} &= \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & d \begin{bmatrix} \frac{\partial \cos(\text{zrot}(\tilde{\mathbf{q}}))}{\partial \mathbf{q}} \\ \frac{\partial \sin(\text{zrot}(\tilde{\mathbf{q}}))}{\partial \mathbf{q}} \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} \omega_w & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & \omega_w & \omega_z & -\omega_y \\ \omega_y & -\omega_z & \omega_w & \omega_x \\ \omega_z & \omega_y & -\omega_x & \omega_w \end{bmatrix} \end{bmatrix} \begin{matrix} \mathbf{x}=\mathbf{x}_{k-1} \\ \mathbf{u}=\begin{bmatrix} d \\ \omega \end{bmatrix} \end{matrix}
 \end{aligned}$$

Table 47 - Pose prediction function and Jacobian matrix

$$\mathbf{X}_k = \begin{bmatrix} f\left(\mathbf{x}_{pose}^{k-1}, \begin{bmatrix} d \\ \omega \end{bmatrix}\right) \\ \mathbf{x}_{pose}^{k-1} \\ \mathbf{x}_{landmark}^1 \\ \mathbf{x}_{landmark}^2 \\ \vdots \end{bmatrix} \quad \mathbf{F}_k = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}} \Big|_{\substack{\mathbf{x}=\mathbf{x}_{pose}^{k-1} \\ \mathbf{u}=\begin{bmatrix} d \\ \omega \end{bmatrix}}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \ddots \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix}$$

Table 48 - Robot and map prediction function and Jacobian matrix

The platform is capable of three different observations. The first is in the change in orientation using the TRX INU. The second is the distance travelled using the wheel encoders. The third is the set of visual features from LK-SURF. The first two observations require both the current and last robot pose to be part of the state. HAR-SLAM already requires the current and last robot pose to create a cross-covariance matrix between the two (see Section 7.1: Modifications of Forgetful SLAM for HAR-SLAM). The third observation requires the current robot location and orientation, as well as the 3D location of each visual feature. The observation of a landmark, using LK-SURF, produces a vector containing the bearing and range from the robot to the landmark. The three observation functions are shown in Table 49. The Jacobian matrices are not shown for the observation functions.

$$\begin{array}{l}
 \mathbf{z}_{encoder} = h_{encoder}(\mathbf{X}) = \left[(p_x^k - p_x^{k-1}) \cos(zrot(q^k)) + (p_y^k - p_y^{k-1}) \sin(zrot(q^k)) \right] \\
 \mathbf{z}_{rotation} = h_{rotation}(\mathbf{X}) = \left[quatMultiply((q^{k-1})^{-1}, q^k) \right] \\
 \mathbf{z}_{feature}^i = h_{feature}^i(\mathbf{X}) = \left[\begin{array}{c} 2 \tan^{-1} \left(\frac{y}{x + \sqrt{x^2 + y^2}} \right) \\ 2 \tan^{-1} \left(\frac{z}{\sqrt{x^2 + y^2} + \sqrt{x^2 + y^2 + z^2}} \right) \\ \sqrt{x^2 + y^2 + z^2} \end{array} \right] \\
 \begin{bmatrix} x \\ y \\ z \end{bmatrix} := \left[(rotM(q^k))^T \left(\mathbf{x}_{landmark}^i - \begin{bmatrix} p_x^k \\ p_y^k \\ 0 \end{bmatrix} \right) \right]
 \end{array}$$

Table 49 - Observation prediction functions

The state and observation models presented in this section are used for all the experimental data shown in this dissertation (see Chapter 5: Robust Kalman Filter and

Chapter 8: Cooperative SLAM with Landmark Promotion). Using a velocity model for the robot allowed the neural net dynamics model to be treated as input control. The velocity model is easy to linearize for error propagation. The input to the neural net was not included in state vector, allowing the neural net to be separated from the state dynamics. The equations presented in this section are unique to the test platform used in this dissertation.

CHAPTER 11: CONCLUSION

11.1. Problem Statement

The main goal of this dissertation was to create a set of algorithms such that a group of inexpensive robots that could cooperatively perform SLAM. A common coordinate system would be associated and used with all the robots. Each robot would fuse various forms of sensor readings, including stereo vision readings, odometry readings, and inertial measurements. The developed SLAM routines should generalize to a group of robots. In addition to the mapping capability, each robot would be targeted at under \$5,000.

11.2. Solution to Problem Statement

The final solution to inexpensive cooperative SLAM was a series of theoretical advances and implementation advances in the area of improved image processing techniques, novel calibration methods, more efficient SLAM techniques, and multi-robot SLAM techniques. New theoretical advances include the Consolidated Unscented Mixed Recursive Filter, the Robust Kalman Filter, Forgetful SLAM, and HAR-SLAM. New implementation advances consist of accurate camera calibration, calibrating a camera rig with an INU, encoder calibration, neural net system modeling, LK-SURF, and LP-SLAM. The completed test platform meets the cost requirements of creating a robot to perform cooperative SLAM at under \$5,000.

11.3. Theoretical Advances

Several advances are presented in this dissertation that relate to Kalman Filters. Four concepts were explored. The first was the Consolidated Unscented Mixed Recursive Filter, which took on the challenge of non-linear systems with non-Gaussian noise. The second was the Robust Kalman Filter, which handles erroneous observations from incorrect feature tracking or matching. The third is Forgetful SLAM, which keeps the Kalman state at a limited size over time. The last is HAR-SLAM, which decouples the Kalman-based SLAM into a chain of smaller Kalman Filters, allowing updates to occur in linear, instead of quadratic time.

The Consolidated Unscented Mixed Recursive Filter is able to handle non-Gaussian noise on non-linear systems. The filter uses the Unscented Transform to handle non-linear systems and does not require a Jacobian to propagate noise. The filter also uses a Gaussian Mixture Model to represent noise, which is more accurate than a single Gaussian representation. The new method works; however, only pieces were used for system modeling, as the method was too computationally intensive. Either the method can be adapted for use in the Robust Kalman Filter or Forgetful SLAM when there is a non-linear system with a difficult Jacobian to calculate, or the system noise needs more accurate modeling.

The Robust Kalman Filter addressed the issue of unexpected observations. Not all sensor noise is Gaussian, but even worse, some noise can be due to algorithm error, or other unpredictable sources. The Robust Kalman Filter can remove outlier observations. An observation needs to agree with the predicted robot motion and the collection of

observations taken at the same time or as a time history. The filter only selects features to apply to the state that pass the X84 rule, which employs the median and median average deviation. The state space is reduced to a single dimension using Principal Component Analysis, making the system quite robust.

A new local SLAM called Forgetful SLAM was created that manages features as long as they are in the current visual field of the robot. This method combines the Robust Kalman Filter with its own state reducing functions to improve and filter both robot positions and the location of visual features. It can be extended to sensor observations, not just visual features. Forgetful SLAM, as its name implies, does not retain features, but it refines observations and the robot state, as well as correlating data in the visual field. Correlation of the data is important, as normally observations and separate landmarks are initialized as independent entities. The correlation that occurs under Forgetful SLAM provides a more accurate representation of the system than the initial set of sensor observations. This accurate representation is used in higher-level (global) SLAM algorithms.

Hierarchical Active Ripple SLAM (HAR-SLAM) was created to handle the massive size of maps, turn maps into chains of states that could be updated in series using Kalman-like update, and handles multiple robot SLAM. It makes some modifications to Forgetful SLAM, which allows it to link forgotten landmarks and states to current state information. The primary mechanism in forming links is the heavy correlation of the state that Forgetful SLAM produces. The correlations make landmarks that are observed again easy to update, closing-the-loop is much easier in HAR-SLAM than current methods like EKF-SLAM, FastSLAM, and GraphSLAM [27]. Multiple robots are joined

with a Robust Kalman Filter layer, which creates a common coordinate system. Common landmarks are then updated using the standard update technique presented in HAR-SLAM.

11.4. Implementation Advances

Several practical advances were offered in this dissertation. One that stands out is LK-SURF, which combines several successful image processing techniques into a spatial and temporal tracker. Several calibration and error modeling techniques were introduced in this dissertation, including camera calibration and Optical INU alignment. Finally, LP-SLAM, which is the practical implementation end of Forgetful SLAM with HAR-SLAM, was presented.

A new feature tracker called LK-SURF was created to perform stereo correspondence, identify a feature, and track it over time reliably. LK-SURF allows each robot to use the inexpensive cameras to range to features accurately and track over time, which simplifies the SLAM problem. Many other researchers exclusively use Harris Corners and use the Lucas-Kanade Tracker, or exclusively use SIFT or SURF features. Harris Corners have no description, and image features perform poorly at picking the exact same feature over time. Combining the descriptive nature of SURF with the temporal tracking ability of the Lucas-Kanade Tracker gave way to an accurate measure of descriptive features.

Several calibration and modeling techniques were presented in this dissertation. Accurate stereo calibration is presented using the OpenCV library. Stereo triangulation

error modeling is performed using synthesized observations from pixel errors. On the Optical INU, an innovative method was developed for aligning the inertial unit with the stereo cameras using a leveled checkerboard and the Kabsch algorithm. The wheel encoders were calibrated using simple techniques, but the dynamics involving the encoders and the control input were modeled using neural nets.

Landmark Promotion SLAM (LP-SLAM) uses HAR-SLAM as a theoretical backend, but limits what information is passed up the various levels of HAR-SLAM, making the system more robust and efficient. LP-SLAM is more of an implementation advance while HAR-SLAM is a theoretical advance. LP-SLAM includes the architecture of how robots could be networked as peer-to-peer nodes, how landmarks are stored spatially, and how landmarks are matched.

11.5. Final Remarks

This dissertation includes several advances in SLAM, from image processing to multi-robot SLAM architectures. LK-SURF is useful in many applications, not just SLAM, the ability to track features in stereo over time and be able to recall features based on SURF descriptors is very powerful. The Robust Kalman Filter is applicable almost anywhere a Kalman Filter is used. In this dissertation, it was used to filter parallel observations robustly, but a time lag would allow the same system to filter over time robustly. Forgetful SLAM, HAR-SLAM, and LP-SLAM all combine to a single useful entity, that allows for linear storage, linear update, and extends to any number of robots.

It is both robust and fast, and does not use brute force techniques like the particle filter or take quadratic time like EKF-SLAM.

The advances presented in this dissertation were all necessary to in order to carry out SLAM as fast as possible on inexpensive platforms with sensors that are much cheaper and less accurate than a LIDAR. This dissertation presents methods that can be extended to other sensors, robot platforms, and have a great capacity to be optimized. The next chapter covers the possibilities of future research that can derive from the work presented in this dissertation.

CHAPTER 12: FUTURE WORK

12.1. Heterogeneous Robots Using the Map

Once a map is built, or as a map is being built by a set of robots, other robots can use the prebuilt map to add sensory information at known locations, or simply navigate a region. Since SURF features are used to form the descriptors, any camera can be used to extract SURF features and compare them to the global map of features. Though stereo cameras are better, a single camera could even work since the positions of the features are known in three dimensions, and a camera provides a bearing to each feature, meaning a single image can result in a full position and orientation estimate. SURF features are robust to size changes, orientation, and slight changes in perspective, making them useful in this application. Both ground and aerial robots could use the map, since the work presented in this dissertation works for 2D and 3D maps. The ASL test platform used in this dissertation maps in 3D, even though it is confined to the ground. The map can be used for complete navigation, with applications such as automating routes inside houses or office buildings. Having an accurate position indoors is useful for autonomous systems that need to be aware of environments.

12.2. Real-time Tracking Using GPU Technology

The SLAM methods presented in this dissertation require relatively low computational power compared to existing methods. However, several methods are

computationally intensive. The image processing is the most intensive component in this dissertation. The Kalman Filters and general linear algebra operations are a burden to the CPU that can be parallelized. LK-SURF tracks hundreds of visible features in stereo; image processing performed in parallel on a Graphics Processing Unit (GPU) would speed up performance, allowing real-time operation. Forgetful SLAM operates on hundreds of visible landmarks; linear algebra performed on the GPU would speed up the performance of the method.

12.2.1. Image Processing on the GPU

Image processing is a computationally intensive process that can be executed in parallel. Extracting SURF features was the most computationally intensive element of the processing in this dissertation work, but this process is also highly parallelizable. The GPU is a computational resource that specializes in parallel processing. Figure 97 shows the architecture of a GeForce 8 GPU by NVIDIA. This architecture contains sufficient of parallel processing power. CUDA is a programming language and structure developed by NVIDIA to use their GPUs for computational processing instead of graphics. OpenCV recently added a GPU version of SURF using CUDA [22]. This is still in the experimental stage; however, the code has been tested and integrated into the ASL Framework. This code provides a real-time version of the Optical INU.

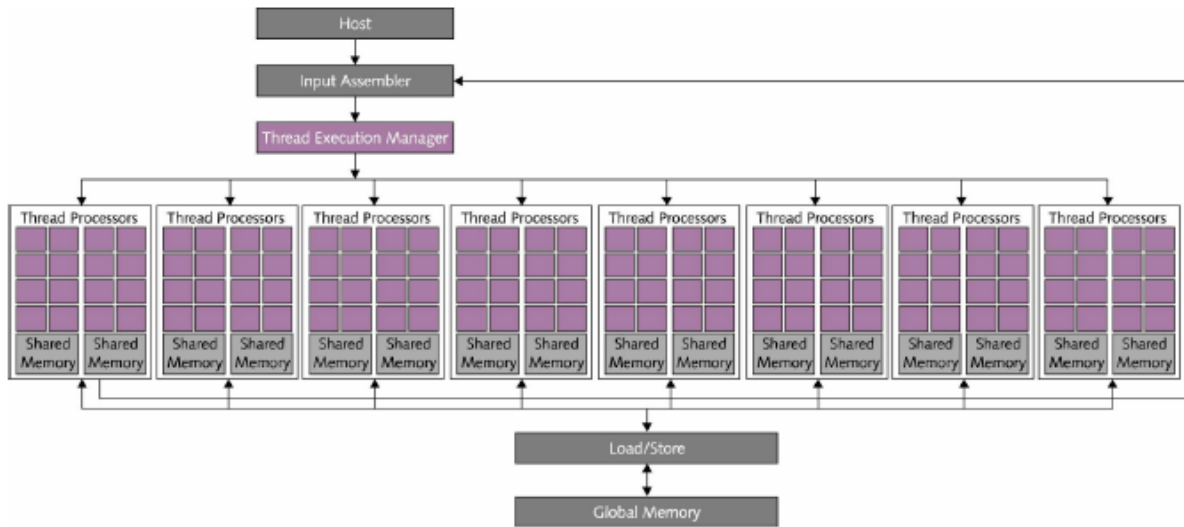


Figure 97 - Architecture of a GeForce 8 GPU [99]

The LK-SURF feature tracker has other computationally intensive components other than SURF feature extraction. Computational intensive functions used in this dissertation include the Lucas-Kanade feature tracker, and the sub-pixel refining routine. Open parallel implementations of these functions are currently not available to for use on the GPU, but implementations should be feasible, making the entire image processing element of this dissertation computationally tractable in real-time.

12.2.2. Linear Algebra on the GPU

Forgetful SLAM can become computationally intensive, especially if hundreds of features are observed at the same time. Matrices become unmanageable, with dimensions well over 300 on each side. Libraries such as CUBLAS are linear algebra libraries that have been ported to CUDA to use the GPU parallel processing[79]. The Cholesky

Decomposition functions, which are used in the Unscented Transform, would benefit significantly from parallel implementation.

HAR-SLAM can be sped up since the entire system is based on linked states, each state could be updated in parallel. Granted, the optimal gains calculated assume ripples, but testing could be done to see if a parallel version would settle to a stable state. If the system does settle, then the parallel version of the system could be sped up from linear computation cost to almost constant computation cost.

12.2.3. Identifying Other Robots in Real-time

If both the image processing and SLAM could be performed in real time given GPU technology, then another element of multiple robots SLAM could be used. The ability to identify another robot in real time would form a link in HAR-SLAM between two robot poses instead of two features. The identification could be done with a much higher level of accuracy, which could result in better map formation. A glyph, such as a checkerboard-like identifying marker, could be attached to each robot that is unique in some way. It would serve two purposes of providing robot identification and a marker that could enable range and relative orientation to be computed using a single camera image. If the pattern of a specific robot can be identified, then a highly reliable association can be made between the two robots. Technically, an observation like that would be the most important and supersede any common landmark sightings.

12.3. Pedestrian Tracking

Robots are generally used for mapping since they are predictable and stable. This is because good motion models can be developed and control inputs are known. However, Figure 98 shows the path of a human that has been put through the Robust Kalman Filter. No model was given, except to limit the possible tracking unit motion was expected to be on the order of how fast a human could move. Using the Robust Kalman Filter, erroneous observations can still be discarded even without a valid motion model. Given that this layer works, technically all layers of LP-SLAM could be applied, and humans could be used to generate a map instead of robots. TRX Systems already locates personnel inside of buildings using human worn sensors; however, the generated maps are inferred from motion rather than sensing the environment.

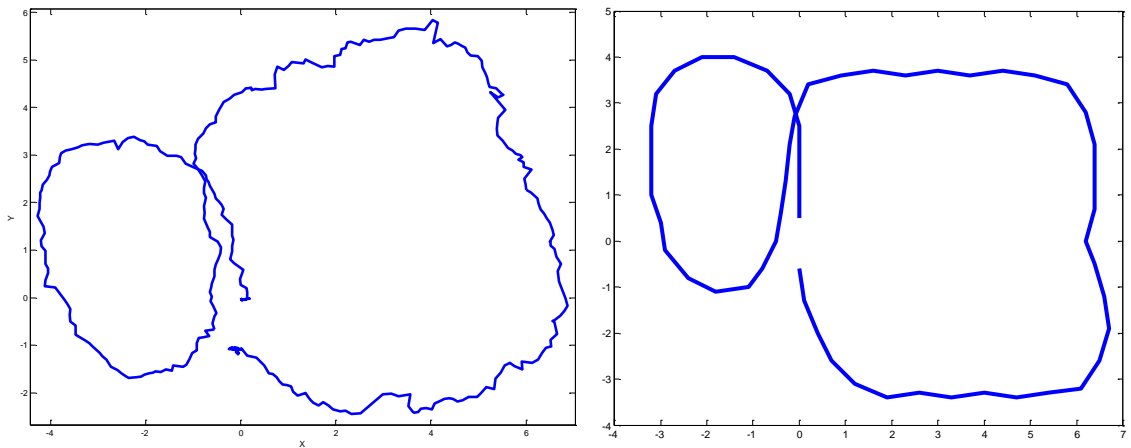


Figure 98 - Figure-eight pedestrian path; Left: Robust Kalman Filter with no model; Right: Path generated by TRX software

12.3.1. Improve Error Models and Model Human Motion

The tests so far have been done without valid motion or error models for humans. This could be corrected using a motion capture system. Simply attaching a motion

capture marker to the Optical INU as shown in Figure 99 would allow software from motion capture companies like Natural Point to provide a complete 6 degree of freedom estimate for the position of the Optical INU. Figure 100 shows a picture of a motion capture camera, and shows a screenshot of capturing the position and orientation of the Optical INU in the capture arena.

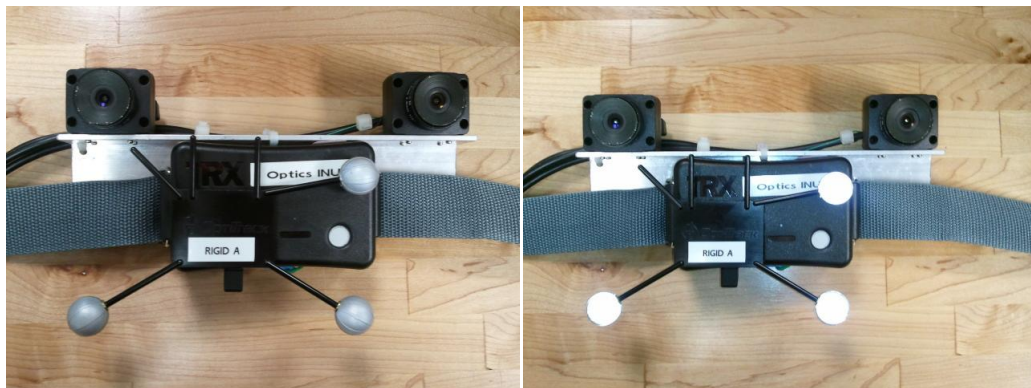


Figure 99 - Motion capture marker on Optical INU. Left: Normal view of the markers; Right: Lit version of the markers, showing the marker reflectivity

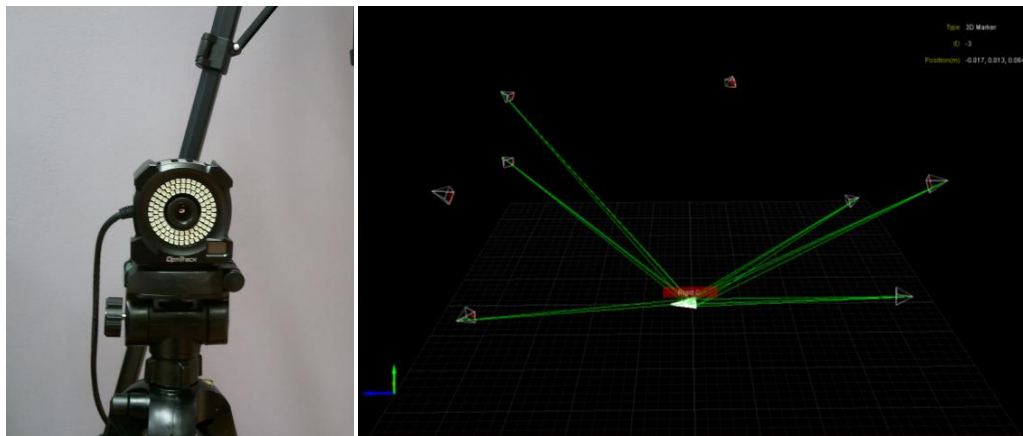


Figure 100 - Left: Motion capture camera from Natural Point; Right: Sample motion capture positioning and orientation estimate using rigid markers

Given accurate ground truth data from a motion capture system, a human motion model could be created using the brute force neural net technique used to model the robot. However, the real control inputs will still be unknown, instead, only a history of sensor values will be used as input to the model. Once we have a model built, we can

estimate noise by comparing untrained data logs of motion capture and compare it to the built neural net model. Other machine learning techniques could be used to produce a human model, such as Hidden Markov Models. A human model is reasonable to obtain, since humans do have limited motion, momentum, and require time to perform actions, thus a model should provide better results than using no model and a giant ball of possible error.

APPENDIX A: USEFUL EMGU EXAMPLES

Stereo Camera Calibration

```
PointF[][] pointSetsRight = FILL FROM N CHECKERBOARDS;
PointF[][] pointSetsLeft = FILL FROM N CHECKERBOARDS;
MCvPoint3D32f[][] allthreeDSets =
    (X,Y) FROM N ACTUAL CHECKERBOARDS (Z)=0;

Emgu.CV.IntrinsicCameraParameters icPRight =
    new Emgu.CV.IntrinsicCameraParameters();
Emgu.CV.IntrinsicCameraParameters icPLeft =
    new Emgu.CV.IntrinsicCameraParameters();
Emgu.CV.ExtrinsicCameraParameters exCP;
Emgu.CV.Matrix<double> fundamentalMatrix;
Emgu.CV.Matrix<double> essentialMatrix;
Emgu.CV.ExtrinsicCameraParameters[] exCPTemp1;
Emgu.CV.ExtrinsicCameraParameters[] exCPTempLeft;

Emgu.CV.CameraCalibration.CalibrateCamera(allthreeDSets,
    pointSetsRight, IMAGE_SIZE, icPRight,
    Emgu.CV.CvEnum.CALIB_TYPE.DEFAULT, out exCPTemp1);

Emgu.CV.CameraCalibration.CalibrateCamera(allthreeDSets,
    pointSetsLeft, IMAGE_SIZE, icPLeft,
    Emgu.CV.CvEnum.CALIB_TYPE.DEFAULT, out exCPTempLeft);

MCvTermCriteria myTermCriteria = new MCvTermCriteria(10000, 0.00001);
Emgu.CV.CameraCalibration.StereoCalibrate(allthreeDSets,
    pointSetsRight, pointSetsLeft, icPRight, icPLeft,
    IMAGE_SIZE,
    Emgu.CV.CvEnum.CALIB_TYPE.DEFAULT, myTermCriteria, out exCP,
    out fundamentalMatrix, out essentialMatrix);
```

Stereo Rectifying

```
Emgu.CV.IntrinsicCameraParameters leftParam = FROM CALIBRATION
Emgu.CV.IntrinsicCameraParameters rightParam = FROM CALIBARTION
Emgu.CV.ExtrinsicCameraParameters extrinsic = FROM CALIBRATION

Matrix<double> rectificationMatrixL = new Matrix<double>(3, 3);
Matrix<double> rectificationMatrixR = new Matrix<double>(3, 3);
Matrix<double> projectionMatrixL = new Matrix<double>(3, 4);
Matrix<double> projectionMatrixR = new Matrix<double>(3, 4);
Matrix<double> dispMapMatrix = new Matrix<double>(4, 4);
Rectangle leftROI = new Rectangle(new Point(0, 0), IMAGE SIZE);
Rectangle rightROI = new Rectangle(new Point(0, 0), IMAGE SIZE);

CvInvoke.cvStereoRectify(
    rightParam.IntrinsicMatrix.Ptr, leftParam.IntrinsicMatrix.Ptr,
    rightParam.DistortionCoeffs.Ptr, leftParam.DistortionCoeffs.Ptr,
    IMAGE SIZE,
    extrinsic.RotationVector.Ptr, extrinsic.TranslationVector.Ptr,
    rectificationMatrixR.Ptr,
    rectificationMatrixL.Ptr,
    projectionMatrixR.Ptr,
    projectionMatrixL.Ptr,
    dispMapMatrix.Ptr,
    Emgu.CV.CvEnum.STEREO_RECTIFY_TYPE.CALIB_ZERO_DISPARITY,
    0, Size.Empty, ref rightROI, ref leftROI);

Matrix<float> LeftMapX = new Matrix<float>(myStereo.ImageSize);
Matrix<float> LeftMapY = new Matrix<float>(myStereo.ImageSize);

CvInvoke.cvInitUndistortRectifyMap(leftParam.IntrinsicMatrix.Ptr,
leftParam.DistortionCoeffs.Ptr, rectificationMatrixL.Ptr,
projectionMatrixL.Ptr, LeftMapX.Ptr, LeftMapY.Ptr);

Matrix<float> RightMapX = new Matrix<float>(myStereo.ImageSize);
Matrix<float> RightMapY = new Matrix<float>(myStereo.ImageSize);

CvInvoke.cvInitUndistortRectifyMap(rightParam.IntrinsicMatrix.Ptr,
rightParam.DistortionCoeffs.Ptr,
rectificationMatrixR.Ptr,
projectionMatrixR.Ptr, RightMapX.Ptr, RightMapY.Ptr);

Image<Bgr, Byte> origLeft = SOME LEFT IMAGE SOURCE
Image<Bgr, Byte> origRight = SOME RIGHT IMAGE SOURCE

Image<Bgr, Byte> newLeft = new Image<Bgr, byte>(myStereo.ImageSize);
Image<Bgr, Byte> newRight = new Image<Bgr, byte>(myStereo.ImageSize);

Emgu.CV.CvInvoke.cvRemap(origLeft.Ptr, newLeft.Ptr, LeftMapX.Ptr,
LeftMapY.Ptr, (int)Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR, new
MCvScalar());

Emgu.CV.CvInvoke.cvRemap(origRight.Ptr, newRight.Ptr, RightMapX.Ptr,
RightMapY.Ptr, (int)Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR, new
MCvScalar());
```

Stereo Triangulation with Epipolar Geometry

Fill private variables from stereo calibration

```
private Matrix FocalLengthLeft = new Matrix(2, 1);
private Matrix FocalLengthRight = new Matrix(2, 1);
private Matrix PrinciplePointLeft = new Matrix(2, 1);
private Matrix PrinciplePointRight = new Matrix(2, 1);
private float SkewLeft = 0;
private float SkewRight = 0;
private Matrix DistortionLeft = new Matrix(5, 1);
private Matrix DistortionRight = new Matrix(5, 1);
private Matrix RotationMatrix = new Matrix(3, 3);
private Matrix Translation = new Matrix(3, 1);

/// <summary>
/// Calculates the 3D points.
/// </summary>
/// <param name="LeftPoint">The left point.</param>
/// <param name="RightPoint">The right point.</param>
/// <returns>3D point</returns>
public Point3D Calculate3DPoint(PointF LeftPoint, PointF RightPoint)
{
    Matrix tempLeftPoint;
    Matrix tempRightPoint;

    Matrix LeftPixel = new Matrix(2, 1);
    Matrix RightPixel = new Matrix(2, 1);

    LeftPixel[0, 0] = LeftPoint.X;
    LeftPixel[1, 0] = LeftPoint.Y;
    RightPixel[0, 0] = RightPoint.X;
    RightPixel[1, 0] = RightPoint.Y;

    Calculate3DPoint(RightPixel, LeftPixel,
        out tempRightPoint, out tempLeftPoint);

    return new Point3D(
        -tempRightPoint[0, 0],
        -tempRightPoint[1, 0],
        tempRightPoint[2, 0]);
}

private void Calculate3DPoint(Matrix MainCameraPixel, Matrix
SecondaryCameraPixel,
    out Matrix MainCameraPoint, out Matrix SecondaryCameraPoint)
{
    //Normalize
    Matrix NewLeft = new Matrix(2, 1);
    NewLeft[0, 0] = (MainCameraPixel[0, 0] - PrinciplePointLeft[0, 0])
        / FocalLengthLeft[0, 0];
    NewLeft[1, 0] = (MainCameraPixel[1, 0] - PrinciplePointLeft[1, 0])
```

```

        / FocalLengthLeft[1, 0];
NewLeft[0, 0] = NewLeft[0, 0] - SkewLeft * NewLeft[1, 0];
NewLeft = comp_distortion_oulo(NewLeft, DistortionLeft);

Matrix NewRight = new Matrix(2, 1);
NewRight[0, 0] = (SecondaryCameraPixel[0, 0] -
    PrinciplePointRight[0, 0]) / FocalLengthRight[0, 0];
NewRight[1, 0] = (SecondaryCameraPixel[1, 0] -
    PrinciplePointRight[1, 0]) / FocalLengthRight[1, 0];
NewRight[0, 0] = NewRight[0, 0] - SkewRight * NewRight[1, 0];
NewRight = comp_distortion_oulo(NewRight, DistortionRight);

Matrix xt = new Matrix(3, 1);
xt[0, 0] = NewLeft[0, 0];
xt[1, 0] = NewLeft[1, 0];
xt[2, 0] = 1;

Matrix xtt = new Matrix(3, 1);
xtt[0, 0] = NewRight[0, 0];
xtt[1, 0] = NewRight[1, 0];
xtt[2, 0] = 1;

//Rotation
Matrix u = RotationMatrix * xt;

//Triangulation
double n_xt2 = xt.DotProduct(xt);
double n_xtt2 = xtt.DotProduct(xtt);

double uDot = u.DotProduct(xtt);
double DD = n_xt2 * n_xtt2 - uDot * uDot;

double dot_uT = u.DotProduct(Translation);
double dot_xttT = xtt.DotProduct(Translation);
double dot_xttu = u.DotProduct(xtt);

double NN1 = dot_xttu * dot_xttT - n_xtt2 * dot_uT;
double NN2 = n_xt2 * dot_xttT - dot_uT * dot_xttu;

double Zt = NN1 / DD;
double Ztt = NN2 / DD;

Matrix X1 = xt * Zt;
Matrix X2 = RotationMatrix.Transpose().Mul(xtt * Ztt -
    Translation);

//Left Points
MainCameraPoint = (X1 + X2) * 0.5;

//Right Points
SecondaryCameraPoint = RotationMatrix * MainCameraPoint +
    Translation;
}

```

```

private Matrix comp_distortion_oulo(Matrix xd, Matrix k)
{
    double k1 = k[0, 0];
    double k2 = k[1, 0];
    double k3 = k[4, 0];
    double p1 = k[2, 0];
    double p2 = k[3, 0];

    Matrix x = xd;

    for (int kk = 1; kk <= 20; kk++)
    {
        double r_2 = x.DotProduct(x);
        double k_radial = 1 + k1 * r_2 + k2 * r_2 * r_2 +
            k3 * r_2 * r_2;
        Matrix delta_x = new Matrix(2, 1);
        delta_x[0, 0] = 2 * p1 * x[0, 0] * x[1, 0] + p2 *
            (r_2 + 2 * x[0, 0] * x[0, 0]);
        delta_x[1, 0] = p1 * (r_2 + 2 * x[1, 0] * x[1, 0]) +
            2 * p2 * x[0, 0] * x[1, 0];
        x = (xd - delta_x).Mul(1 / k_radial);
    }

    return x;
}

```

APPENDIX B: MATLAB IMPLEMENTATIONS OF VARIOUS KALMAN FILTERS

Basic Kalman Filter

```
function [Xnew Pnew] = KalmanFilter (Xold, Pold, Z, F, B, H, U, Q, R)
%KALMANFILTER Basic Kalman Filter.
% KALMANFILTER(Xold, Pold, Z, F, B, H, U, Q, R), for known parameters
% Xold = Previous Estimated State
% Pold = Previous Estimated State Covariance
% Z = Current Observation
% F = State Function Matrix
% B = Control Matrix
% H = Observation Matrix
% U = Current Control Input
% Q = State Noise Covariance Martix
% R = Observation Noise Covariance Matrix
%
%
% See also EXTENDEDKALMANFILTER, UNSCENTEDKALMANFILTER.
%
% Copyright 2010 John Karvounis.

%% Predict Stage
Xpred = F * Xold + B * U;
Ppred = F * Pold * F' + Q;

%% Innovation Stage
Y = Z - H * Xpred;
S = H * Ppred * H' + R;

%% Kalman Gain
K = (Ppred * H') / S;

%% Update Stage
Xnew = Xpred + K*Y;
Pnew = (eye(size(K,1),size(H,2)) - K * H) * Ppred;
```

Extended Kalman Filter

```
function [Xnew Pnew] = ExtendedKalmanFilter
    (Xold, Pold, Z, f, h, U, Q, R)
%EXTENDEDKALMANFILTER Extended Kalman Filter.
%   EXTENDEDKALMANFILTER(Xold, Pold, Z, f, h, U, Q, R), for known
%   system parameters
%   Xold = Previous Estimated State
%   Pold = Previous Estimated State Covariance
%   Z = Current Observation
%   f = System Function, of the form Xnew = @f(Xold, U)
%   h = Observation Function, of the form Y = @h(X)
%   U = Current Control Input
%   Q = State Noise Covariance Matrix
%   R = Observation Noise Covariance Matrix
%
%   See also KALMANFILTER, UNSCENTEDKALMANFILTER.

%   Copyright 2010 John Karvounis.

%% Linearization of F
syms x_1_1;
X_Sym = x_1_1*zeros(size(Xold));
for r = 1:size(Xold,1)
    for c = 1:size(Xold,2)
        eval(['syms x_' num2str(r) '_' num2str(c) ';']);
        eval(['X_Sym(r,c) = x_' num2str(r) '_' num2str(c) ';']);
    end
end

syms u_1_1;
U_Sym = u_1_1*zeros(size(U));
for r = 1:size(U,1)
    for c = 1:size(U,2)
        eval(['syms u_' num2str(r) '_' num2str(c) ';']);
        eval(['U_Sym(r,c) = u_' num2str(r) '_' num2str(c) ';']);
    end
end

F_Sym = feval(f, X_Sym, U_Sym);
F_Jacobian = jacobian(F_Sym, X_Sym);
F = subs(subs(F_Jacobian, X_Sym, Xold), U_Sym, U);

%% Predict Stage
Xpred = feval(f, Xold, U);
Ppred = F * Pold * F' + Q;
```

```

%% Linearization of H
syms x_1_1;
X_Sym = x_1_1*zeros(size(Xold));
for r = 1:size(Xold,1)
    for c = 1:size(Xold,2)
        eval(['syms x_' num2str(r) '_' num2str(c) ';']);
        eval(['X_Sym(r,c) = x_' num2str(r) '_' num2str(c) ';']);
    end
end

H_Sym = feval(h, X_Sym);
H_Jacobian = jacobian(H_Sym, X_Sym);
H = subs(H_Jacobian, X_Sym, Xpred);

%% Innovation Stage
Y = Z - feval(h, Xpred);
S = H * Ppred * H' + R;

%% Kalman Gain
K = (Ppred * H') / S;

%% Update Stage
Xnew = Xpred + K*Y;
Pnew = (eye(size(K,1),size(H,2)) - K * H) * Ppred;

```


Unscented Kalman Filter

```
function [Xnew Pnew] = UnscentedKalmanFilter
    (Xold, Pold, Z, f, h, U, Q, R)
%UNSCENTEDKALMANFILTER Unscented Kalman Filter.
% UNSCENTEDKALMANFILTER(Xold, Pold, Z, f, h, U, Q, R), for known
% system parameters
% Xold = Previous Estimated State
% Pold = Previous Estimated State Covariance
% Z = Current Observation
% f = System Function, of the form Xnew = @f(Xold, U)
% h = Observation Function, of the form Y = @h(X)
% U = Current Control Input
% Q = State Noise Covariance Matrix
% R = Observation Noise Covariance Matrix
%
% See also KALMANFILTER, EXTENDEDKALMANFILTER.

% Copyright 2010 John Karvounis.

%% Calculate Parameters
L = length(Xold);
alpha = 0.5;
beta = 2;
kappa = 0;

%% Calculate Sigma Points
Sigma1 = CalculateSigmaPoints(Xold, Pold, L, alpha, beta, kappa);

%% Predict Stage
Sigma2 = cell(length(1+2*L),1);
for i = 1:(1+2*L)
    Sigma2{i} = feval(f, Sigma1{i}, U);
end

[Xpred, Ppred] = CalculateSigmaProperties(Sigma2, L, alpha, beta,
kappa);
Ppred = Ppred + Q;

%% Calculate Sigma Points (Again)
Sigma3 = CalculateSigmaPoints(Xpred, Ppred, L, alpha, beta, kappa);

%% Innovation Stage
Sigma4 = cell(length(1+2*L),1);
for i = 1:(1+2*L)
    Sigma4{i} = feval(h, Sigma3{i});
end

[Zpred, S] = CalculateSigmaProperties(Sigma4, L, alpha, beta, kappa);
S = S + R;

Pxz = CalculateSigmaCrossCovariance(Xpred, Sigma3, Zpred, Sigma4, L,
alpha, beta, kappa);
```

```

%% Kalman Gain
K = Pxz / S;

%% Update Stage
Xnew = Xpred + K * (Z - Zpred);
Pnew = Ppred - K * S * K';
end

function Sigma = CalculateSigmaPoints (X, P, L, alpha, beta, kappa)

%% Calculate Parameters
lambda = alpha * alpha * (L + kappa) - L;
[cholRoot isNotPos] = chol(P, 'lower');
%% Calculate Sigma Points
if (~isNotPos)
    root = sqrt((L + lambda)) * cholRoot;
else
    root = norm(P)*eye(size(P));
end

Sigma = cell(2*L+1,1);
Sigma{1} = X;
for i = 1:L
    Sigma{i+1} = X + root(:,i);
end
for i = 1:L
    Sigma{i+1+L} = X - root(:,L-i+1);
end
end

function [X P] = CalculateSigmaProperties(Sigma, L, alpha, beta, kappa)

%% Calculate Parameters
lambda = alpha * alpha * (L + kappa) - L;

%% Calculate Mean
X = (lambda / (L + lambda)) * Sigma{1};
for i = 2:(1 + 2 * L)
    X = X + (1 / (2 * (L + lambda))) * Sigma{i};
end

%% Calculate Covariance
P = ((1 - alpha * alpha + beta) + lambda / (L + lambda)) ...
    * (Sigma{1} - X) * (Sigma{1} - X)';
for i = 2:(1 + 2 * L)
    P = P + (1 / (2 * (L + lambda))) ...
        * (Sigma{i} - X) * (Sigma{i} - X)';
end
end

```

```

function Pxz = CalculateSigmaCrossCovariance
    (Xmean, SigmaX, Zmean, SigmaZ, L, alpha, beta, kappa)

%% Calculate Parameters
lambda = alpha * alpha * (L + kappa) - L;

%% Calculate Cross-Covariance
Pxz = ((1 - alpha * alpha + beta) + lambda / (L + lambda)) ...
    * (SigmaX{1} - Xmean) * (SigmaZ{1} - Zmean)';
for i = 2:(1 + 2 * L)
    Pxz = Pxz + (1 / (2 * (L + lambda))) ...
        * (SigmaX{i} - Xmean) * (SigmaZ{i} - Zmean)';
end
end

```

Gaussian Consolidation

```
function [X P] = CalculateMergedSigmaProperties
    (Sigma1, w1, Sigma2, w2, L, alpha, beta, kappa)

%% Calculate Parameters
lambda = alpha * alpha * (L + kappa) - L;
a1 = w1 / (w1 + w2);
a2 = w2 / (w1 + w2);

%% Calculate Mean
X = a1 * (lambda / (L + lambda)) * Sigma1{1} + ...
    a2 * (lambda / (L + lambda)) * Sigma2{1};
for i = 2:(1 + 2 * L)
    X = X + ...
        a1 * (1 / (2 * (L + lambda))) * Sigma1{i} + ...
        a2 * (1 / (2 * (L + lambda))) * Sigma2{i};
end

%% Calculate Covariance
P = 0;
for i = 2:(1 + 2 * L)
    P = P + ...
        a1 * (1 / (2 * L)) *
            (sqrt(L / (L + lambda)) *
                (Sigma1{i} - Sigma1{1}) + Sigma1{1} - X) *
            (sqrt(L / (L + lambda)) *
                (Sigma1{i} - Sigma1{1}) + Sigma1{1} - X)' +
        a2 * (1 / (2 * L)) *
            (sqrt(L / (L + lambda)) *
                (Sigma2{i} - Sigma2{1}) + Sigma2{1} - X) *
            (sqrt(L / (L + lambda)) *
                (Sigma2{i} - Sigma2{1}) + Sigma2{1} - X)';
end
```

LIST OF REFERENCES

- [1] Hugh Durrant-Whyte and Time Bailey, "Simultaneous Localization and Mapping: Part I," *Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99-110, June 2006.
- [2] S. Thrun, D. Fox, and W. Burgard, "A probabilistic approach to concurrent mapping and localization formobile robotics," *Machine Learning*, p. 31, 1998.
- [3] K. S. Chong and L. Kleeman, "Feature-based mapping in real, large scale environments using an ultrasonic array," *International Journal of Robotics Research*, pp. 3-19, January 1999.
- [4] P. Jensfelt, D. Kragic, J. Folkesson, and M. Björkman, "A Framework for Vision Based Bearing Only 3D SLAM," in *Internation Conference on Robotics and Automation*, 2006, p. 1564–1570.
- [5] Anastasios I. Mourikis, Stergios I. Roumeliotis, and Joel W. Burdick, "SC-KF Mobile Robot Localization: A Stochastic Cloning Kalman Filter for Processing Relative-State Measurements," in *Transactions on Robotics*, 2007, pp. 717-730.
- [6] Hugh F. Durrant-Whyte, "Uncertain Geometry in Robotics," *IEEE Journal of Robotics and Automation*, pp. 23-31, 1988.
- [7] Guilherme N. DeSouza and Avinash C. Kak, "Vision for Mobile Robot Navigation: A Survey," in *Transactions on Pattern Analysis and Machine Intelligence*, 2002, pp. 237-267.
- [8] Thomas Lemaire and Simon Lacroix, "SLAM with Panoramic Vision," *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 91-111, February 2007.
- [9] B. Funk, Method and System for Locating and Monitoring First Responders, 2008.
- [10] A. Milella and R Siegwart, "Stereo-Based Ego-Motion Estimation Using Pixel Tracking and Iterative Closest Point," in *International Conference on Computer Vision Systems*, 2006, pp. 21-27.
- [11] C. G. Harris, *Geometry from visual motion*, A. Blake and A. Yuille, Eds. Cambridge, MA: MIT Press, 1992.

- [12] David D. Diel, "Stochastic Constraints for Vision-Aided Inertial Navigation," Cambridge, MA, Masters Thesis 2005.
- [13] Ronen Lerner, Ehud Rivlin, and Ilan Shimshoni, "Landmark Selection for Task-Oriented Navigation," in *Transactions on Robotics*, vol. 23, 2007, pp. 494-505.
- [14] Anthony Mallet, Simon Lacroix, and Laurent Gallo, "Position Estimation in Outdoor Environments using Pixel Tracking and Stereovision," in *International Conference on Robotics & Automation*, San Francisco, CA, 2000, pp. 3519-3524.
- [15] TRX Systems, Inc. (2009) Firefighter Sentinel System. [Online]. <http://www.trxsystems.com>
- [16] Goksel Dedeoglu, Maja J. Mataric, and Gaurav S. Sukhatme, "Incremental online topological map building with a mobile robot," in *Mobile Robots XIV*, vol. 3838, Boston, MA, 1999.
- [17] R. Jürgens, T. Boß, and W. Becker, "Estimation of self-turning in the dark: comparison between active and passive rotation," *Experimental Brain Research*, vol. 128, no. 4, pp. 491-504, 1999.
- [18] James J. Gibson, "The Visual Perception of Objective Motion and Subjective Movement," *Psychological Review*, vol. 61, no. 5, pp. 304-314, September 1954.
- [19] Ian P Howard and Gang Hu, "Visually induced reorientation illusions," *Perception*, vol. 30, pp. 583-600, 2001.
- [20] J. Lobo, C. Queiroz., and J. Dias, "World feature detection using stereo vision and inertial sensors," *Robotics and Autonomous System*, no. 44, pp. 69-81, 2003.
- [21] Navid Nourani-Vatani, Jonathan Roberts, and Mandyam V. Srinivasan, "IMU Aided 3D Visual Odometry for Car-Like Vehicles," in *Australasian Conference on Robotics and Automation*, 2008.
- [22] (2009) Intel Open Source Computer Vision Library. [Online]. <http://opencv.willowgarage.com/>
- [23] Chris Harris and Mike Stephens, "A Combined Corner and Edge Detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147-151.
- [24] Jean-Yves Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the algorithm," OpenCV Document 2000.
- [25] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding*, vol. 110, no. 3,

- pp. 346-359, 2008.
- [26] David D. Diel, Paul DeBitetto, and Seth Teller, "Epipolar Constraints for Vision-Aided Inertial Navigation," in *Workshop on Motion and Vision Computing*, vol. 2, 2005, pp. 221-228.
- [27] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.
- [28] Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba, "A Solution to the Simultaneous Localization and Map Building (SLAM) Problem," in *Transactions on Robotics and Automation*, vol. 17, 2001.
- [29] S. Siiksakulchai, S. Thongchai, D. M. Wilkes, and K. Kawamura, "Mobile Robot Localization using an Electronic Compass for Corridor Environment," in *IEEE Conference on Systems, Man, and Cybernetics*, vol. 5, Nashville, TN, 2000, pp. 3354-3359.
- [30] Jonghyuk Kim and Salah Sukkarieh, "Improving the Real-Time Efficiency of Inertial SLAM and Understanding its Observability," in *International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004, pp. 21-26.
- [31] Pantelis Elinas, Robert Sim, and James J. Little, " σ SLAM: Stereo Vision SLAM Using the Rao-Blackwellised Particle Filter and a Novel Mixture Proposal Distribution," in *International Conference on Robotics and Automation*, Orlando, Florida, 2006, pp. 1564-1570.
- [32] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problem," *ASME—Journal of Basic Engineering*, vol. 82, pp. 35-45, 1960.
- [33] Vincenzo Lippiello, Bruno Siciliano, and Luigi Villani, "Position-Based Visual Servoing in Industrial Multirobot Cells Using a Hybrid Camera Configuration," in *Transactions on Robotics*, 2007, pp. 73-86.
- [34] Robert Sim, Pantelis Elinas, Matt Griffin, and James J. Little, "Vision-based SLAM using the Rao-Blackwellised Particle Filter," in *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2005.
- [35] A.J. Davison and N. Kita, "3D Simultaneous Localisation and Map-Building Using Active Vision for a Robot Moving on Undulating Terrain," in *Computer Vision and Pattern Recognition*, 2001.
- [36] T. Lemaire, S. Lacroix, and J. Solà, "A practical 3D Bearing-Only SLAM algorithm," in *International Conference on Intelligent Robots and Systems*, 2005, pp. 2449- 2454.

- [37] Andrew J. Davison, Yolanda Gonzalez Cid, and Nobuyuki Kita, "Real-Time 3D SLAM with Wide-Angle Vision," in *Proceedings of the IFAC Symposium on Intelligent Autonomous Vehicles*, Lisbon, 2004.
- [38] Andrew J. Davison, "Real-Time Simultaneous Localisation and Mapping with a Single Camera," in *Ninth IEEE International Conference on Computer Vision*, Nice, France, 2003, pp. 1403-1410.
- [39] S. Takezawa, D. C. Herath, and G. Dissanayake, "SLAM in Indoor Environments with Stereo Vision," in *International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004, pp. 1867-1872.
- [40] Jae-Hean Kim and Myung Jin Chung, "SLAM with Omni-directional Stereo Vision Sensor," in *International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003, pp. 442-447.
- [41] Yi Ma, Jana Kosecka, and Shankar S. Sastry, "Vision Guided Navigation for a Nonholonomic Mobile Robot," in *Transactions on Robotics and Automation*, 1999, pp. 521-536.
- [42] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix, "Vision-Based SLAM: Stereo and Monocular Approaches," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 343-364, 2007.
- [43] Jaime Valls Miro, Gamini Dissanayake, and Weizhen Zhou, "Vision-based SLAM using natural features in indoor environments," in *Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2005, pp. 151-156.
- [44] Miguel Angel Garcia and Agusti Solanas, "3D Simultaneous Localization and Modeling from Stereo Vision," in *International Conference on Robotics and Automation*, New Orleans, 2004, pp. 847-853.
- [45] Fredrik Orderud, "Comparison of Kalman Filter Estimation Approaches for State Space Models with Nonlinear Measurements," in *Scandinavian Conference on Simulation and Modeling*, 2005.
- [46] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse, "MonoSLAM: Real-Time Single Camera SLAM," in *Conference on Pattern Analysis and Machine Intelligence*, vol. 29, 2007.
- [47] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit, "FastSLAM 2.0 An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Acapulco, 2003.

- [48] P. Newman, "On the Structure and Solution of the Simultaneous Localisation and Map Building Problem," Sydney, PhD Thesis 2000.
- [49] E.A. Wan and R. Van Der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000*, Lake Louise, Alta, 2000, pp. 153-158.
- [50] Tim Bailey and Hugh Durrant-Whyte, "Simultaneous Localization and Mapping (SLAM): Part II," *Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108-117, September 2006.
- [51] Sebastian Thrun and Michael Montemerlo, "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *International Journal on Robotics Research*, vol. 5, no. 6, pp. 403-430, 2005.
- [52] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," in *Transactions on Robotics*, vol. 23, 2007, pp. 34-46.
- [53] Michael Montemerlo, Sebastian Thrun, and William Whittaker, "Conditional Particle Filters for Robot Localization and People Tracking," in *International Conference on Robotics and Automation*, 2002, pp. 695-701.
- [54] R. Sim, P. Elinas, M. Griffin, and J. J. Little, "Vision-based slam using the rao-blackwellised particle filter," in *Workshop on Reasoning with Uncertainty in Robotics*, Edinburgh, 2005.
- [55] Tim K. Marks, Andrew Howard, Max Bajracharya, Garrison W. Cottrell, and Larry Matthies, "Gamma-SLAM - Using Stereo Vision and Variance Grid Maps for SLAM in Unstructured Environments," in *International Conference on Robotics and Automation*, 2007.
- [56] Pantelis Elinas and James J. Little, "Stereo vision SLAM: Near real-time learning of 3D point-landmark and 2D occupancy-grid maps using particle filters," in *Visual SLAM Workshop*, San Diego, CA, 2007.
- [57] David Nistér, Oleg Naroditsky, and James Bergen, "Visual Odometry for Ground Vehicle Applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3-20, 2006.
- [58] Simon Thompson and Satoshi Kagami, "Revising Stereo Vision Maps in Particle Filter Based SLAM using Localisation Con dence and Sample History," in *International Conference on Autonomous Robots and Agents*, Palmerston North, New Zealand, 2004, pp. 218-223.

- [59] David G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Seventh IEEE International Conference on Computer Vision*, vol. 2, Kerkyra, Greece, 1999, pp. 1150-1157.
- [60] Stephen Se, David G. Lowe, and James J. Little, "Vision-Based Global Localization and Mapping for Mobile Robots," in *Transactions on Robotics*, 2005, pp. 364-375.
- [61] Howie Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113-126, 2001.
- [62] Maxim A. Batalin and Gaurav S. Sukhatme, "Design and Analysis of an Efficient Local Algorithm for Coverage and Exploration," in *Transactions on Robotics*, vol. 23, 2007, pp. 661-675.
- [63] Marius Kloetzer and Calin Belta, "Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions," in *Transactions on Robotics*, vol. 23, 2007, pp. 320-330.
- [64] Justin Clark and Rafael Fierro, "Cooperative Hybrid Control of Robotic Sensors for Perimeter Detection and Tracking," in *American Control Conference*, 2005, pp. 3500-3505.
- [65] Michael M. Zavlanos and George J. Pappas, "Potential Fields for Maintaining Connectivity of Mobile Networks," in *Transactions on Robotics*, 2007, pp. 812-816.
- [66] Meng Ji and Magnus Egerstedt, "Distributed Coordination Control of Multiagent Systems While Preserving Connectedness," in *Transactions on Robotics*, vol. 23, 2007.
- [67] Jorge Cortes and Francesco Bullo, "Coordination and Geometric Optimization via Distributed Dynamical Systems," *SIAM Journal on Control and Optimization*, vol. 44, no. 5, pp. 1543-1574, 2006.
- [68] Anurag Ganguli, Jorge Cortes, and Francesco Bullo, "On Rendezvous for Visually-Guided Agents in a Nonconvex Polygon," in *Decision and Control*, 2005, pp. 5686-5691.
- [69] General Dynamic Robotic Systems. (2009) GDRS - Robotics. [Online]. <http://www.gdrs.com/robotics/>
- [70] Motilal Agrawal and Kurt Konolige, "Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS," in *International Conference on Pattern Recognition*, Hong Kong, 2006, pp. 1063-1068.

- [71] Kat Bradley and Kaylin Spitz, "Face Tracking and Pose Approximation," Computing Research Association, 2009.
- [72] Akitsugu Noguchi and Keiji Yanai, "A SURF-based Spatio-Temporal Feature for Feature-fusion-based Action Recognition," in *Third Workshop on HUMAN MOTION Understanding, Modeling, Capture and Animation*, Crete, 2010.
- [73] Frank R. Hampel, Elvezio M. Ronchetti, Peter J. Rousseeuw, and Werner A. Stahel, *Robust Statistics: The Approach Based on Influence Functions*. New York, USA: Wiley-Interscience, 1986.
- [74] T. Tommasini, A. Fusiello, E. Trucco, and V. Roberto, "Making Good Features Track Better," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Santa Barbara, 1998, pp. 178 - 183.
- [75] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer-Verlag, 2002.
- [76] Matthew Turk and Alex Pentland, "Face Recognition Using Eigenfaces," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71-86, Winter 1991.
- [77] Dimitri P Bertsekas, *Dynamic Programming and Optimal Control*, Third Edition ed. Belmont, Massachusetts: Athena Scientific, 2005.
- [78] Galen R Shorack and Jon A Wellner, *Empirical Process with Applications to Statistics*. Seattle: Society for Industrial and Applied Mathematics, 2009.
- [79] NVIDIA. (2009) CUDA Zone. [Online].
http://www.nvidia.com/object/cuda_home.html
- [80] P C Mahalanobis, "On the generalised distance in statistics," in *Proceedings of the National Institute of Sciences of India*, 1936, pp. 49-55.
- [81] Arthur Gregory, Ming Lin, Stefan Gottschalk, and Russell Taylor, "A Framework for Fast and Accurate Collision Detection for Haptic Interaction," in *IEEE Proceedings of Virtual Reality*, Houston, 1999, pp. 38-45.
- [82] M Teschner, B Heidelberger, M Mueller, D Pomeranets, and M Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," in *Proceedings of Vision, Modeling, Visualization*, 2003, pp. 47-54.
- [83] Michael Stanley, "Implementation of Kalman Filter to Tracking Custom Four-Wheel Drive Four-Wheel-Steering Robotic Platform," University of Maryland, College Park, Masters Thesis 2010.

- [84] A. Bandyopadhyay, System and Method for Determining Location of Personnel and/or Assets Both Indoors and outdoors Viw Map Generation and/or Map Matching Techniques, 2008.
- [85] B. Funk, A. Bandyopadhyay, and E. Kohn, Method and System for First Resonders, 2007.
- [86] (2009) Freifunk. [Online]. <http://start.freifunk.net/>
- [87] Jared Napora, "Implementation, Evaluation, and Applications of Mobile Mesh Networks for Platforms in Motion," Univerity of Maryland, College Park, Masters Thesis 2009.
- [88] (2011) AForge.NET. [Online]. <http://www.aforgenet.com/>
- [89] (2011) Roborealm vision for machines. [Online]. <http://www.roborealm.com/>
- [90] Microsoft. (2011, March) Microsoft Robotics Studio. [Online]. <http://www.microsoft.com/robotics/>
- [91] Princeton University. (2011, March) Princeton Autonomous Vehicle Engineering - Prosepect 12. [Online]. <http://pave.princeton.edu/about/projects/prospect12>
- [92] Point Grey Research, FlyCapture2, 2011.
- [93] Microsoft. (2011, March) App Hub: Develop for Windows Phone & XBox. [Online]. <http://create.msdn.com>
- [94] Mathworks, Matlab 7.9, 2009.
- [95] Wolfgang Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Crystallographica*, vol. 34, pp. 827-828, September 1978.
- [96] Shailesh Lakkad, "Modeling and Simulation of Steering Systems for Autonomous Vehicles," Tallahassee, FL , Master Thesis 2004.
- [97] Danwei Wang and Feng Qi, "Trajectory Planning for a Four-Wheel-Steering Vehicle," in *International Conference on Robotics & Automation*, Seoul, Korea, 2001, pp. 3320-3325.
- [98] Kishan Mehrotra, Chilukuri Mohan, and Sanjay Ranka, *Elements of Artificial Neural Nets*, 2nd ed. Caimbridge, Massachusetts: MIT Press, 1997.
- [99] Tom R. Halfhill, "Parallel Processing With CUDA: Nvidia's High-Performance

Computing Platform Uses Massive Multithreading," *Microprocessor: The Insider's Guide to Microprocessor Hardware*, 2008.