MULTIPLE INTEGRATED NAVIGATION SENSORS FOR IMPROVED OCCUPANCY GRID FASTSLAM

THESIS

Christopher P. Weyers, Second Lieutenant, USAF

AFIT/GCE/ENG/11-08

AFIT/GCE/ENG/11-08

MULTIPLE INTEGRATED NAVIGATION SENSORS FOR IMPROVED OCCUPANCY GRID FASTSLAM

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Christopher P. Weyers, BSCSE

Second Lieutenant, USAF

March 2011

AFIT/GCE/ENG/11-08

MULTIPLE INTEGRATED NAVIGATION SENSORS
FOR IMPROVED OCCUPANCY GRID FASTSLAM

Christopher P. Weyers, BSCSE
Second Lieutenant, USAF

Approved:

//signed//	March 2011
Gilbert L. Peterson (Chairman)	Date
//signed//	March 2011
John F. Raquet (Member)	Date
//signed//	March 2011
Kenneth A. Fisher (Member)	Date

AFIT/GCE/ENG/11-08

Abstract

An autonomous vehicle must accurately observe its location within the environment to interact with objects and accomplish its mission. When its environment is unknown, the vehicle must construct a map detailing its surroundings while using it to maintain an accurate location. Such a vehicle is faced with the circularly defined Simultaneous Localization and Mapping (SLAM) problem. However difficult, SLAM is a critical component of autonomous vehicle exploration with applications to search and rescue. To current knowledge, this research presents the first SLAM solution to integrate stereo cameras, inertial measurements, and vehicle odometry into a Multiple Integrated Navigation Sensor (MINS) path. The implementation combines the MINS path with LIDAR to observe and map the environment using the FastSLAM algorithm. In real-world tests, a mobile ground vehicle equipped with these sensors completed a 140 meter loop around indoor hallways. This SLAM solution produces a path that closes the loop and remains within 1 meter of truth, reducing the error 92% from an image-inertial navigation system and 79% from odometry FastSLAM.

AFIT/GCE/ENG/11-08

To my family, who unknowingly carried me through each success and each failure.

Acknowledgements

First of all, I would like to thank my advisor for having patience with me in learning and for believing in my abilities before I could. I would like to express my appreciation to committee members for providing a much needed perspective from their backgrounds. I am also deeply grateful to the staff of the Advanced Navigation Technology Center without whose extensive help, I would be inexplicably lost. The support of the Sensors Directorate has been crucial as the sponsor for this research.

I would also like to thank my group members for their dependable teamwork in class, and the rest of my computer and electrical engineering classmates for their support. Lastly, I must thank my housemate and lab-mates for their presence and company as we each labored through our difficulties. You have all made this great opportunity worthwhile.

Christopher P. Weyers

Table of Contents

List of Figures

List of Tables

List of Symbols

List of Abbreviations

MULTIPLE INTEGRATED NAVIGATION SENSORS
FOR IMPROVED OCCUPANCY GRID FASTSLAM

I. Introduction

As the environment of warfare shifts from open fields to urban cities, it becomes increasingly difficult and dangerous for human forces to occupy and control the battlefield. Recent increases in vehicle autonomy have allowed the warfighter to take a step away from the danger. Pilots and soldiers now extensively control Remotely Piloted Aircraft (RPA) and Explosive Ordinance Disposal (EOD) robots to remove themselves from some of the most vulnerable situations [2].

While their use is undisputedly saving lives, these existing systems often require a large team to operate and lack the autonomy that is often associated with robots [2]. Increasing autonomy in these vehicles would alleviate the need for such a large support team. Reducing the number of people involved allows the military to deploy more vehicles where they are needed and require less human operators and maintainers.

Additional situations in future operations also benefit from the use of autonomous vehicles. Existing platforms are incapable of target identification or Combat Search and Rescue (CSAR) missions in environments that demand ground transportation. An urban area CSAR mission requires that a vehicle maintain an accurate position of itself while navigating around obstacles to its eventual goal.

To increase the autonomy of these vehicles, they must be able to navigate without the direct control of a driver. Each of these applications requires the vehicle to observe its environment and store a representation to pinpoint the location of a target within the environment and act on it. Whether the vehicle must report on an adversary, handle explosives, or rescue a casualty, it needs to use a map to accomplish its mission. In addition to external sensors, this requires a precise navigation position.

The warfighter of today navigates almost exclusively through the use of the Global Positioning System (GPS) network. Since the advent of the GPS satellite constellation, GPS receivers have afforded the military an invaluable resource for precise navigation. This technology has a vast number of applications for both military and commercial users, and has enabled wildly accurate global navigation.

However, this capability has also created a significant dependence on accurate navigation. Over time, many systems and many people have come to rely on GPS to accomplish their mission. Without a signal, not only would many drivers end up lost, but many military weapon systems would also be unable to function.

Unfortunately, although the GPS signal is global, it is not available everywhere. In bad weather, inside most building, between tall structures, and underground, a GPS signal is unreliable and often too attenuated to be usable. An additional consideration is the adversaries ability to jam the GPS signal and render it unusable. Because of the number of capabilities now defined by their accuracy, the military needs other means to achieve similar precision navigation in all operating environments.

1.1 Motivation

A vehicle operating in areas without GPS requires a navigation method that functions without using external signals. This means the vehicle must rely on its own sensors to explore an environment while maintaining an accurate position. Current missions demand an accurate position to always be available, especially when a vehicle relays that location to its own operators or to an allied vehicle.

If the concept of internal navigation sounds unlikely, consider comparing current technology to biological systems. Humans and other animals are able to navigate in both known and unknown environments with incredible accuracy. From a design standpoint, biological navigation amounts chiefly to stereo vision coupled with touch and inertial sensors.

Since the goal is to obtain a similar level of navigation, consider sensor configurations attempts at mimicking the abilities of animals. That is, using cameras to see and an inertial unit to sense movement. Combine these sensors and the result is an artificial navigation vehicle. When external signals are unavailable due to the physical environment or the actions of adversaries, the scenario demands the use of local sensors. An autonomous vehicle of this kind can be used to explore and collect information or even take action if necessary. Even in hostile environments, an internally navigating vehicle can accomplish its mission without relying on vulnerable external means.

1.2 Problem Definition

In mobile robotics, the Simultaneous Localization and Mapping (SLAM) problem addresses where a vehicle is and what its environment contains [60]. A functional SLAM solution is a key component of an autonomous vehicle, especially one whose mission involves gaining knowledge of unknown areas. It provides the ability to explore unknown environments without prior knowledge and report on its findings. Because of the sensors involved, the solution does not rely on any external signals commonly used for navigation or communication.

The SLAM problem is often represented as a Dynamic Bayes Network (DBN) from time 0 to time t consisting of the vehicle state, control inputs u_t, measurement observations z_t, and external landmarks θ [46]. Figure 1 illustrates their relationship in a DBN.

At time 0, the vehicle maintains $\mathbf{x}_0$ and makes observation z_0, which corresponds

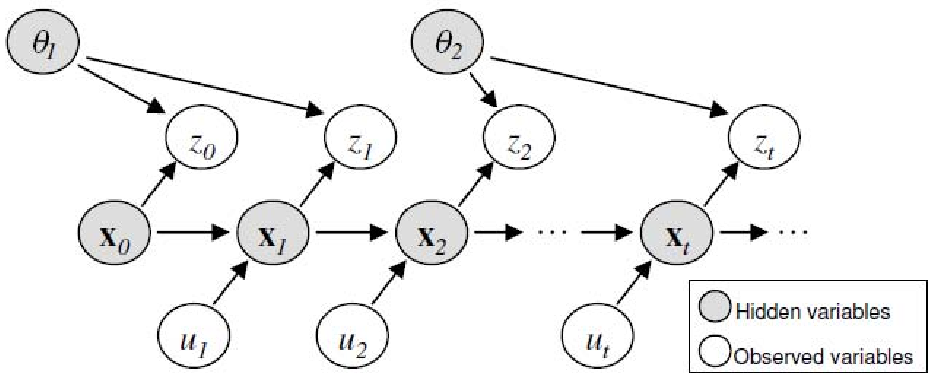

Figure 1. The SLAM problem as a dynamic Bayes network. Vehicle states $\mathbf{x}_{0:t}$, controls $u_{1:t}$, and observations $z_{0:t}$, have subscripts denoting time, with states $\mathbf{x}_{0:t}$ highlighted to show conditional independence. Landmarks θ are numbered arbitrarily [8, 46].

to a physical landmark θ_1. The vehicle is then given a control command u_1, which results in a new state $\mathbf{x}_1$ where z_1 observes the same landmark. After receiving u_2, the new vehicle observation z_2 corresponds to a different landmark θ_2. In this way, u_t results in new positions and z_t can observe previous landmarks in addition to new ones.

This DBN arrangement results in an important property of the network. Because each node $\mathbf{x}_1$ to $\mathbf{x}_t$ of the vehicle path $\mathbf{x}_{0:t}$ is a parent to each observation $z_{0:t}$, the observations are independent from each other given the vehicle path and physical landmark locations [52]. Algorithms make use of this independence property when producing SLAM solutions by using only the current path node and separating observed landmarks [46].

The DBN also illustrates another important aspect of the SLAM problem. State $\mathbf{x}_t$ increments with u_t and z_t, depending only on $\mathbf{x}_{t-1}$ and external landmarks θ. Thus, the illustration makes the Markov assumption that all relevant information is captured in the current state [52]. A SLAM algorithm that makes this assumption is called an online solution [61] and only uses the current u_t and z_t. This research is focused on online solutions, treating the problem as illustrated in Figure 1, and maintaining a current belief at each time t, an important property of a mapping vehicle.

Alternatively, algorithms that store previous measurement and state data to calculate together are offline solutions. They do not make the Markov assumption and thus store all observed data and state information. Therefore, offline solutions require more memory then online solutions and make calculations once all data has been gathered [61].

SLAM is often described fundamentally as a paradoxical problem. In order for a vehicle to answer what an environment looks like, it must know where it is relative to its surroundings. Likewise, in order to answer the question of where it is, the vehicle must know its environment to place itself. This description presents the SLAM problem to require an iterative solution. To remain an online solution, a probabilistic method accounts for errors and compensates in an attempt to minimize them.

Many current SLAM algorithms use two sensors to take measurements. The first uses encoders on the motors and wheels to measure its position. This is called vehicle odometry, and due to its nature, it produces a path subject to compounding error the solution seeks to reduce. The second sensor is any that measures ranges from the vehicle to objects in the environment. These errors do not compound as they are relative to the vehicle. Because of this disparity in measurement accuracy, the most common source of error is that in the path.

A straightforward SLAM implementation approaches the problem by measuring position, then applying ranges to this position. In this approach, path errors become map errors when incorporating range observations and go uncorrected. This strategy of not correcting position only satisfies half of the fundamental SLAM description and is called a loose coupling of the sensors. For accuracy, the SLAM problem demands more tightly coupled sensors in a solution.

1.3 Research Goals

Previous researchers have already developed an internal navigation system using vision and inertial data without mapping [65]. One arrangement consists of stereo cameras tightly coupled with an inertial sensor to produce an accurate position on many different platforms in real-time [23]. From here on, it is called the Fletcher-Veth Scale-Invariant Feature Transform (FV-SIFT), named after its developers and the SIFT feature tracking algorithm [40].

It uses a Kalman filtering method to achieve a functional path. Specific situations adversely affect the system due to its dependence on good images and smooth movement to achieve its path [64]. A wheeled ground vehicle can mitigate such problems by using inputs from the vehicle itself, but this previous system does not incorporate additional input sensors.

The goal of this research is to introduce two dimensional mapping capability to FV-SIFT. A vehicle carries the operating navigation hardware and a novel solution incorporates the inputs from stereo cameras, an inertial sensor, and vehicle odometry into a single path.

This work requires any mobile robotic vehicle that has the required computational and sensory equipment. This includes all FV-SIFT hardware, a laser range finder, and wheeled odometry. The indoor hallway environment provides real test data comparable to popular data sets from recent research [29].

1.4 Vision Navigation & Mapping

This research combines the sensory inputs of two prominent strategies into a single solution. It makes use of the odometry and laser ranges of the vehicle with stereo images and inertial measurements of the FV-SIFT hardware into a two dimensional mapping algorithm. The solution combines motion from integrated inertial data, extracted from stereo camera images, and measured by odometry together in a linear Kalman filter to create a single path. A particle filter then uses that path with laser ranges to produce a map of the traveled environment.

This research builds on the earlier work that developed FV-SIFT with the addition of mapping functionality to improve its navigation capabilities. This work also supports current research in SLAM by providing a method to combine inertial, optical (camera), and mechanical (odometry) sensors into a single solution and apply known mapping techniques for multiple inputs.

With more sensors to estimate a position, this research produced a more accurate path with 92% less error than FV-SIFT and 79% less error than mapping with vehicle odometry in conducted tests. The paths compare to a building floor plan since the vehicle operator knows the approximate path taken. The operator measures the accuracy of a path by determining the distance from known locations. Figure 2 displays a floor plan of the hallway environment with the approximate true path taken. Certain points have been surveyed to serve as the reference points for this true path.

Unfortunately, the operator cannot know exact truth values at every point, as existing external systems like GPS are unavailable in the testing environment. This serves to reinforce the purpose of the research and illustrates the need for internal navigation.

In addition to an accurate path, the solution also constructs a map of the environment traveled. It implements a particle filtering technique to further adjust the path. The algorithm incorporates laser range scans to observe obstacles such as walls or doorways and place them in the map. The overall strategy maintains many different maps that represent different beliefs of the environment. The particle filter incorporates the second half of the SLAM problem description by duplicating maps with the

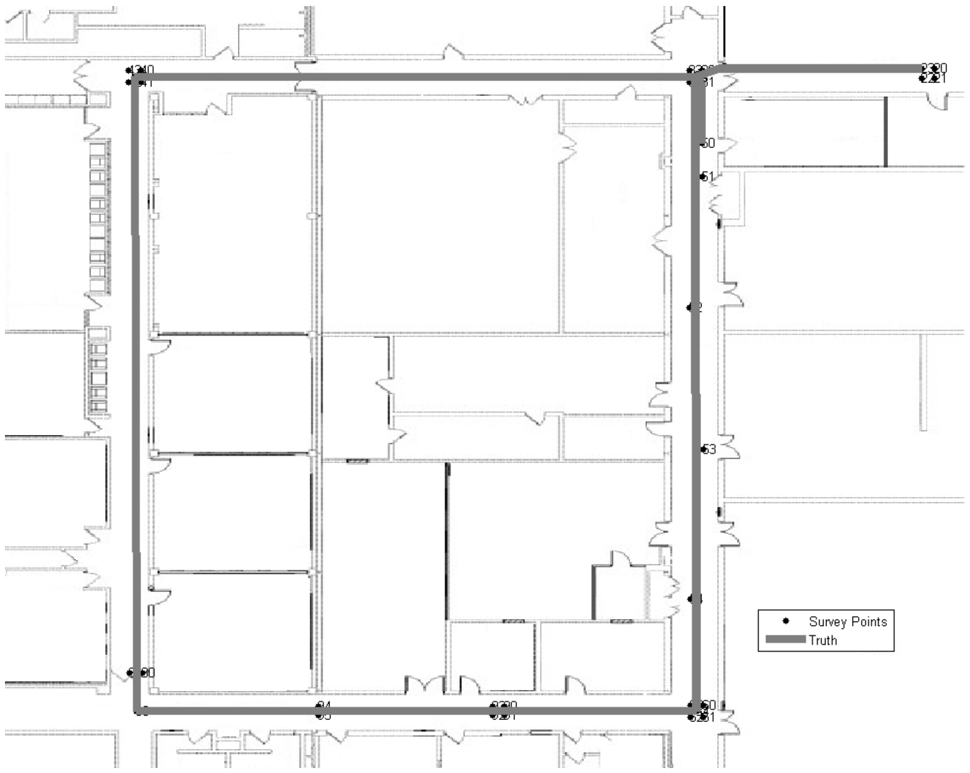

Figure 2. A floor plan of the building environment with surveyed locations. The desired path of an accurate solution is the approximate truth looping around the hallways.

highest chances of being correct and removing ones inconsistent with measurements.

1.5 Research Applications & Summary

A vehicle equipped with this system is capable of independently exploring an unknown location and reporting on that environment. It can return from an exploration mission having produced a current, accurate map of the area. This map would provide decision makers with the most current information for immediate planning decisions.

Such a ground vehicle provides new search and rescue capabilities in multiple environments. The implementation presented only discusses testing indoors, but the same solution applies in any relatively flat area. Future research will likely extend SLAM capabilities to three dimensions, building an even more complete picture of the unknown environment.

An autonomous ground vehicle would be able to explore dangerous locations where the safety of personnel warrants the use of unmanned vehicles. It allows operators the ability to remove themselves from potentially hazardous situations and lowers the risk to human life.

This research also applies to environments where GPS is unavailable either temporarily or permanently. Deploying a map-building autonomous vehicle gives its operators a platform that does not rely on GPS or external signals to explore unknown areas of interest. In this way, a map can serve to measure distances and navigate without requiring external communication.

The following chapter provides a background on work related to this topic that has been researched or demonstrated previously. Chapter 3 presents details of the algorithms and implementations used in the research to produce paths and maps from sensor inputs. Chapter 4 describes the specific parameters used, data collected, results of this research, and comparison to alternate methods. The final chapter comments on what can be done to further this research area with potential topics to explore.

II. Background & Related Work

This work builds on existing algorithms for using camera images and other sensors for navigation and mapping. In addition to the field of robotic mapping, this includes the fields of vehicle navigation, image processing, and machine vision. Recent research in each of these areas has resulted in significant progress. This research extends this work by combining these research areas.

The metholodology presented in Chapter 3 uses most but not all of the concepts, techniques, and strategies of this chapter. The implementation combines odometry, a camera path, and inertial measurements in a linear Kalman filter that produces the path necessary for localization. It also implements a particle filter that uses the path and a range sensor to build a grid map of the environment. This chapter explains the map representation and the particle filter separately.

The first sections of this chapter describe different strategies for storing the map and representing position. This is followed by foundations in localization algorithms as the first component of the SLAM problem. Next, this chapter discusses mapping algorithms and more recent methods to improve them. Following the presentation of machine vision algorithms used to generate and track image features, the core of image navigation solutions are discussed. Finally, this chapter covers techniques used to construct maps from inputs gathered by camera hardware.

2.1 Representing Maps

A common decision a SLAM solution must make is how to represent the environment internally. The map itself is the most important result, as this serves as both the most visible output and the environment for later navigation. Although this chapter discusses more advanced strategies later, there are two basic ways to represent a map of the environment; metric and topological maps.

One method to represent the map is metric, where objects are stored by their absolute position. This is commonly done by maintaining an occupancy grid of which locations are open and which contain obstacles. Grid squares may have centimeter resolution, which adds up when map sizes approach the kilometer range. Storing a global metric map such as this carries a significant memory requirement which can affect computation as much as complexity, so several algorithms have been proposed to reduce this requirement [46].

Instead of the common metric map, another strategy is the topological map representation. One solution takes an approach similar to early research but changes its map representation [12]. This means that landmarks are placed based on their location relative to others. This difference in maps continues to be a divided area, with solutions using both methods of representation.

This research uses an occupancy grid metric map, as displayed in several successful implementations [28, 29]. A metric map is easy to interpret and measure, and the grid arrangement allows a straightforward implementation on a computer [33].

Even with a fast algorithm, any solution must represent and update its map, especially if one is maintained for each particle as presented in this research. Because larger maps demand more particles for consistency, this memory requirement grows, and is most often the most significant contribution to computation time.

2.2 Representing Positions

To represent vehicle position, both a location and an orientation are needed. A two dimensional coordinate system requires three variables, whereas a three dimensional system requires six variables. In matrix representation, six variables means a $6^2 = 36$ element matrix multiplication, so keeping the dimension low reduces the data requirements to $3^2 = 9$ elements.

A two dimensional position consists of an x and a y coordinate. Because orientation is desired, the vehicle must also track a heading angle θ. Together, these three elements make up a vehicle pose. A two dimensional pose is represented by vector $\mathbf{s}$ in Equation 1.

$$\mathbf{s} \equiv \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1}$$

Due to the geometric nature of the pose definition, the two dimensional trigonometry of the pose determines their addition and subtraction. To add a pose $\mathbf{s}^a$ to another $\mathbf{s}^b$, the result is not straightforward vector addition but instead defined by the product of a rotation matrix in Equation 2.

$$\mathbf{s}^a + \mathbf{s}^b \equiv \begin{bmatrix} x^a \\ y^a \\ \theta^a \end{bmatrix} + \begin{bmatrix} \cos(\theta^a) & -\sin(\theta^a) & 0 \\ \sin(\theta^a) & \cos(\theta^a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^b \\ y^b \\ \theta^b \end{bmatrix} \tag{2}$$

For notation purposes, pose superscripts transfer to its elements. In the same manner as addition, subtraction of a second pose $\mathbf{s}^b$ from an initial pose $\mathbf{s}^a$ uses another rotation matrix product as expressed in Equation 3.

$$\mathbf{s}^a - \mathbf{s}^b \equiv \begin{bmatrix} \cos(\theta^b) & \sin(\theta^b) & 0 \\ -\sin(\theta^b) & \cos(\theta^b) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^a - x^b \\ y^a - y^b \\ \theta^a - \theta^b \end{bmatrix} \tag{3}$$

In subsequent equations, all pose additions and subtractions use their respective operation from Equations 2 and 3. This pose definition allows for easy communication with the environment as represented in a two dimensional grid. This research leverages this useful property as it applies to the occupancy grid for mapping.

2.3 Vehicle Localization

Vehicle localization focuses on determining where a vehicle is within a defined area. It is often difficult for humans to do this given a blueprint of a building or a map of a cave, and autonomous vehicles have many of the same problems. The most prevalent solutions approach this problem from a probabilistic point of view [15]. Probabilistic localization algorithms redefine the localization solution to be the probability that a vehicle occupies any given location.

To localize in an environment at time t, the current vehicle pose $\mathbf{s}_t$ is combined with all measurements $z_{0:t}$ as a Bayesian filtering problem [15]. The filter represents an estimate of the posterior probability $p(\mathbf{s}_t|z_{0:t})$ at each time t. With each new time t, it updates $\mathbf{s}_t$ with a motion model discussed later in this chapter, then an update phase finds the posterior by the Bayes theorem:

$$p(\mathbf{s}_t|z_{0:t}) = \frac{p(z_t|\mathbf{s}_t)\ p(\mathbf{s}_t|z_{0:t-1})}{p(\mathbf{s}_t|z_{0:t-1})}. \tag{4}$$

In this way, the sensor measurements are incorporated to obtain a pose estimate. It is important to note that the above calculation is only possible under the assumption that z_t is conditionally independent of all previous observations $z_{0:t-1}$ [52].

For comparison, consider how humans navigate a building with a map but without a location marked. One would have to use their senses to narrow down a location; if eyesight is unavailable, the situation is the same for the blind. Related research details how the same concepts discussed for vehicles can successfully be used to create a solution humans can use for themselves [31].

For these strategies, localization requires previous knowledge of the environment, such as the floor plan of a building or map of a park. If designing a vehicle to operate in another location, it needs a map for the new environment to localize to. If a current map of the building or area exists, this would not be possible, because there is no way of knowing the correctness of a provided map. A solution can only assume it is accurate and result in a false solution if the map has errors.

2.4 Simultaneous Localization & Mapping

Because localization suffers from any inaccuracies in the map provided, most research extends the problem to include map building [57]. The challenge is therefore to perform Concurrent Mapping and Localization (CML), more commonly called the Simultaneous Localization and Mapping (SLAM) problem [39].

Perhaps the most difficult aspect of SLAM is its cyclical solution. In order for a vehicle to build a map of its surroundings, an accurate position is required. However, as stated in the last section, localization is only possible when a map is available [60]. This explains why solutions have only been presented recently, as they rely on probabilistic methods combined with accurate hardware and fast computation [18]. This has been the case for early methods and more recent algorithms alike [58].

SLAM solutions iterate by first estimating a pose within an existing map, and then updating the map to incorporate new measurements. This is the general approach taken after an early solution proved that a SLAM solution both exists and converges [16]. Its estimation approach to representing vehicle state is a principle that nearly all subsequent methods use.

The SLAM posterior at time t is similar to the localization posterior, but includes more variables. SLAM requires the current pose $\mathbf{s}_t$, all measurements $z_{0:t}$, the map Θ, and all control inputs $u_{1:t}$. Equation 5 shows the posterior calculation as that of a Bayes filter, where η is a normalization constant.

$$p(\mathbf{s}_t, \Theta | z_{0:t}, u_{1:t}) = \eta \; p(z_t | \mathbf{s}_t, \Theta) \int p(\mathbf{s}_t | \mathbf{s}_{t-1}, u_t) p(\mathbf{s}_{t-1}, \Theta | z_{0:t-1}, u_{1:t-1}) d\mathbf{s}_{t-1} \tag{5}$$

The posterior typically includes a term for data association, but this research removes it due to differences in map representation. The data association problem requires the algorithm to correctly map each observation z_t to each landmark $\theta \in \Theta$ for an accurate solution. Because this research saves Θ as a metric map rather than a set of landmarks, this association can be considered part of the measurement model presented later.

Because evaluating the integral in Equation 5 is computationally prohibitive, SLAM solutions use statistical estimating techniques to approximate the filter [45]. A common approach to estimate Equation 5 uses an Extended Kalman Filter (EKF), which linearizes differences in vehicle motion to estimate a current pose and an associated error [16]. The primary issue with an EKF algorithm is its K^2 matrix storage requirement for an area with K landmarks, with up to a $O(K^3)$ time complexity for matrix operations.

This does not scale well in areas that produce large maps if many landmarks are used. Concerned about the consistency of results, a newer algorithm was introduced to reduce EKF linearization errors, producing a slightly more accurate estimate [11]. Other algorithms attempt to further reduce this error by using an UKF to produce better results [19].

Some critical examination of the filters finds limitations to the linearization as well as the tendency for the estimate to become overly optimistic [3]. This results in less error than needed, resulting in an increasingly inaccurate estimate. This flaw is present regardless of whether an EKF, an iterated version of the EKF, or a UKF is used. Neither improvement makes any change to the complexity from the $K \times K$

error matrix in a solution with K landmarks.

An improved EKF SLAM solution uses incremental matrix factorization [36]. It seeks to keep the information matrix in an upper triangular form and the useful data together, reducing the effective complexity. The reduced computation allowed for a better map when the vehicle returned to the same location multiple times, closing the loop in the path.

Some EKF based SLAM implementations take many hours to compute a simple vehicle path on modern machines. Simply due to its quadratic running time, this removes any considerations of a real-time implementation. For this reason, the EKF strategy for landmark mapping is abandoned in this research for a different approach using a particle filter.

2.4.1 Particle Filtering.

To take a different approach to the SLAM problem, recall its DBN representation in Figure 1. Its structure allows each state to represent all information at the current time t, and each observation z_t and control u_t to be conditionally independent [52].

Because of the conditional probabilities of the DBN, the integral in Equation 5 can be rewritten as the product shown in Equation 6 [52]. This factors the SLAM

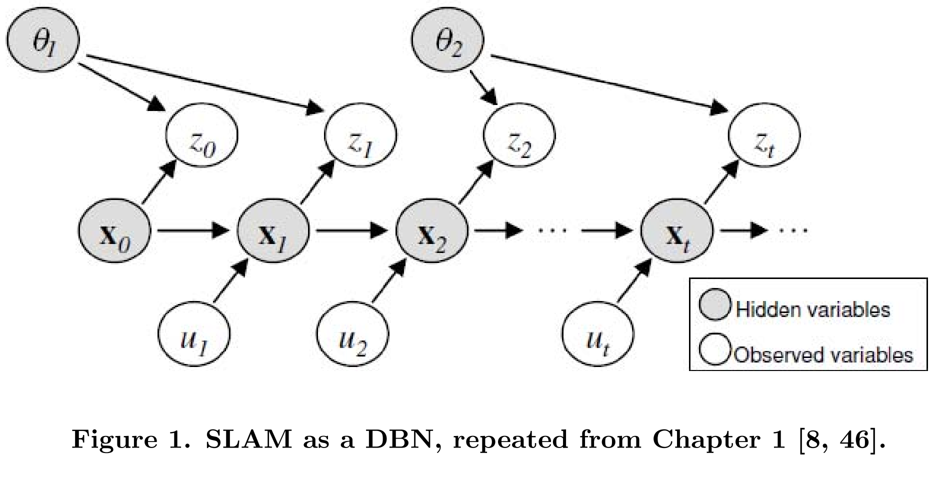

Figure 1. SLAM as a DBN, repeated from Chapter 1 [8, 46].

posterior into one pose posterior and K landmark estimates [45].

$$p(\mathbf{s}_t, \Theta | z_{0:t}, u_{1:t}) = p(\mathbf{s}_t | z_t, u_t) \prod_{k=1}^{K} p(\theta_k | \mathbf{s}_t, z_{0:t}, u_{1:t}) \tag{6}$$

A different approach to approximating this posterior is to use a particle filter. A particle filter maintains the error distribution through many samples, or particles, each storing a pose estimate and landmark locations. This is in contrast to the large error matrix in an EKF. Instead of maintaining one estimate with covariances, many particles each keep one estimate, creating a discrete distribution of the desired error.

Sampling M independent particles $[m]$ from the posterior results in an empirical estimate given in Equation 7, where δ is the Dirac delta function at location $\{\mathbf{s}^{[m]}, \Theta^{[m]}\}$ [17]. This sampling removes the conditional probabilities on z and u, while covering the desired distribution by the law of large numbers.

$$p(\mathbf{s}, \Theta | z, u) = \frac{1}{M} \sum_{m=1}^{M} \delta_{\{\mathbf{s}^{[m]}, \Theta^{[m]}\}} \; d\mathbf{s} \; d\Theta \tag{7}$$

The very first presentation of the particle filter notes its application to SLAM, performing better than exact inference [17]. The basic strategy to include many particles $[m]$ works well initially, but each time error is added, the distribution spreads further apart and becomes less accurate [61]. This potential complication is mitigated through resampling, the key aspect of this filter. Resampling affects the distribution by deleting lowly weighted (high error) particles and copying highly weighted (low error) ones in a similar manner to genetic algorithms.

The specific filter discussed in this work is a Rao-Blackwellized Particle Filter (RBPF), named after the factorization method of marginalizing out variables. Using this factorization with resampling reduces the number of particles required to maintain the desired distribution. The RBPF maintains the set P of M particles to be distributed across the location posterior [17]. Each step of the RBPF is listed in Algorithm 1 for each time t.

Algorithm 1 Generic RBPF Operation

```
for all times $t$ do
    // sequential importance sampling
    for all particles $[m] \in P$ do
        Sample $r_t^{[m]}$ from a simple distribution
        Include $r_t^{[m]}$ in the set of previous samples
        Calculate and normalize the importance weight $w_t^{[m]}$
    end for
    // selection step
    Reproduce $r_t^{[m]}$ with high $w_t^{[m]}$, and suppress $r_t^{[m]}$ with low $w_t^{[m]}$
    $M$ samples are now distributed along the posterior
    // markov chain monte carlo
    Apply transition probabilities to obtain each $r_t^{[m]}$
end for
```

A RBPF first performs Sequential Importance Sampling (SIS) by drawing from a Gaussian proposal distribution. It then calculates an importance weight $w_t^{[m]}$ for each particle $[m]$ and uses a Sampling Importance Resampling (SIR) algorithm to select a new set of samples in Figure 3 [45]. In terms of the SLAM posterior, $w_t^{[m]}$ is the ratio of distributions at each sample location as shown in Equation 8.

$$w_t^{[m]} = \frac{\textit{target distribution}}{\textit{proposal distribution}} = \frac{p(\mathbf{s}_t^{[m]}|z_t, u_t)}{p(\mathbf{s}_t^{[m]}|z_{t-1}, u_t)} \tag{8}$$

Researchers soon applied the RBPF to SLAM seeking to reduce the computational complexity of the EKF sensor update [46]. This method is appropriately named Fast-SLAM and achieves a running time of $O(M \log K)$ for M particles and K landmarks. This makes it able to handle much larger environments and still be feasibly computational. The FastSLAM algorithm completes three parts for each particle in each time step t [45]. This process is shown with resampling in Algorithm 2.

To map larger environments, the strategy simply increases the number of particles

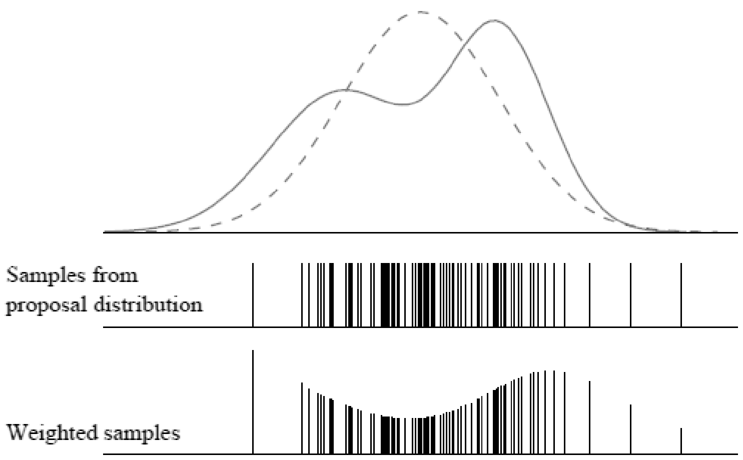

Figure 3. An example of sampling and weighting. The algorithm draws from a proposal Gaussian distribution (dashed line), then weights each sample to achieve the target posterior distribution (solid line) [45].

Algorithm 2 Basic FastSLAM Procedure

```
for all times $t$ do
  for all particles $[m] \in P$ do
    Sample new pose $\mathbf{s}_t^{[m]}$ given control $u_t$ // motion model
    Update landmarks corresponding to observation $z_t$ // measurement model
    Assign weight $w_t^{[m]}$ to each $[m]$ // weighting
  end for
  Resample $P$ according to each weight $w_t^{[m]}$
end for
```

in the RBPF to produce a consistently accurate map. FastSLAM is not without fault, as a large number of particles slows computation time. Some analysis further explains that increasing the map size will eventually cause any number of particles to become inconsistent [4].

FastSLAM therefore contains a balance between speed and quality, dependent on how many particles are used. Although an improved version incorporates a smaller particle distribution closer to the measurement, it still must determine a required number of particles [47]. Even as they show the second version converges with a single particle, note that many particles are still required for a consistent practical implementation. Its quality also depends greatly on the accuracy of the sensors and models within it. Regardless, FastSLAM remains a scalable implementation.

One problem with RBPF solutions is that a large enough path results in measurement errors outside the particle distribution. This results in a map unable to close large loops and perhaps intersecting its path at an incorrect location. Other algorithms have been presented to combat this effect, one method using distributed particles within a different filter [20]. It has a complicated running time and requires a comparable number of particles, yet produces a robust map. A later publication improves the algorithm complexity to a level not greater than a localization solution [21].

Another SLAM solution, GraphSLAM, seeks to use past measurement and path data to update the map, instead of just the most recent [62]. By representing the posterior as a graph of nodes, this algorithm performs better in large-scale environments. However, because it uses previous data, it is a full SLAM solution where a map can only be computed after all measurements have been taken. This is useful for building a reference map at a later time, but cannot compute a map as the vehicle explores the environment.

The authoritative book on the subject reviews both versions of FastSLAM and other popular methods in great detail [61]. Its authors compare each method using various data sets and present their results, finding that all SLAM methods provide different advantages but none have demonstrated themselves to be better than FastSLAM in all situations. As such, FastSLAM provides the foundation for the implementation used in this research.

2.4.2 Filter Models.

As described, FastSLAM is not a single algorithm; rather, it is a framework for solving the SLAM problem. It specifies each of its steps in terms of input and effect and than the implementation details. These steps are generally separate, and replaceable with different system models [61].

Because the resampling process is dictated by the RBPF, FastSLAM requires two models. The first is the motion model, to incorporate a probabilistic pose distribution using control input u_t, and the second is the measurement model, to update the map and weight this distribution using observation z_t.

2.4.2.1 Motion Model.

Of particular note is the development of a motion model for vehicle odometry [61]. This model bases vehicle motion upon the mechanics of a wheeled ground vehicle as is commonly used for mapping. Odometry records poses directly at a measurement interval, denoted by different times t. During this time, the vehicle moves to its current pose $\mathbf{s}_t$ from previous pose $\mathbf{s}_{t-1}$. Equation 9 depicts the difference between poses $\delta\mathbf{s}$ of interest in the motion model.

$$\delta\mathbf{s} = \mathbf{s}_t - \mathbf{s}_{t-1} = \begin{bmatrix} \delta x \\ \delta y \\ \delta\theta \end{bmatrix} \tag{9}$$

The purpose of the motion model is to represent this movement from time $t-1$ to t and record an updated location and heading. Figure 4 shows an example of pose difference $\delta\mathbf{s}$ graphically.

The motion model describes $\delta\mathbf{s}$ as a rotation from the old pose to the new one, a translation toward it, and second rotation to the new heading [61]. This allows the model to add Gaussian noise to the rotation and translation components rather than a multidimensional oval. Equation 10 define how the three motion components are calculated from $\delta\mathbf{s}$.

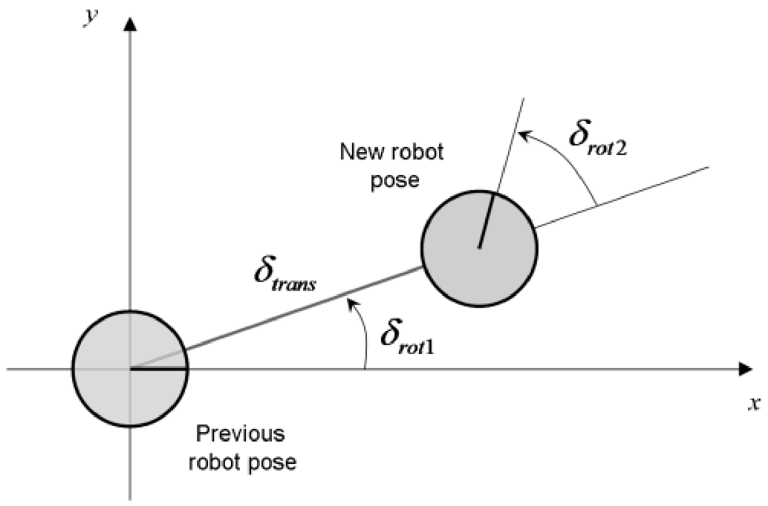

Figure 4. The odometry motion model expresses movement as rotation δ_{rot1} toward a new pose, translation δ_{trans} in that direction, and rotation δ_{rot2} to a new heading [61].

$$\begin{bmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{bmatrix} = \begin{bmatrix} \left|\text{atan2}(\delta y,\ \delta x) - \theta\right| \\ \sqrt{\delta x^2 + \delta y^2} \\ \left|\delta\theta - \delta_{rot1}\right| \end{bmatrix} \tag{10}$$

Since these parameters are treated as a measurement, they remain the same for each particle. To represent the distribution, four parameters α represent the expected error between various movement. Each particle then uses each α value to calculate the error applied to its new pose. This is shown in Equation 11, where $\mathcal{N}$ samples a zero-mean normalized Gaussian distribution and multiplication is element-wise.

$$\begin{bmatrix} d_{rot1} \\ d_{trans} \\ d_{rot2} \end{bmatrix} = \begin{bmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{bmatrix} - \begin{bmatrix} \mathcal{N} \\ \mathcal{N} \\ \mathcal{N} \end{bmatrix} \begin{bmatrix} \alpha_1|\delta_{rot1}| + \alpha_2|\delta_{trans}| \\ \alpha_3|\delta_{trans}| + \alpha_4|\delta_{rot1} + \delta_{rot2}| \\ \alpha_1|\delta_{rot2}| + \alpha_2|\delta_{trans}| \end{bmatrix} \tag{11}$$

Each particle $[m]$ then adds the sampled difference to its previous pose $\mathbf{s}_{t-1}^{[m]}$ to obtain its new $\mathbf{s}_t^{[m]}$ in Equation 12. This results in particle set P being distributed to represent the desired pose error. The goal of the particle filter is that one particle retains the correct vehicle path.

$$\mathbf{s}_t^{[m]} = \mathbf{s}_{t-1}^{[m]} + \begin{bmatrix} d_{trans}^{[m]} \cos(d_{rot1}^{[m]}) \\ d_{trans}^{[m]} \sin(d_{rot1}^{[m]}) \\ \left| d_{rot1}^{[m]} + d_{rot2}^{[m]} + \theta^{[m]} \right| \end{bmatrix} \tag{12}$$

All together, the motion model propagates vehicle pose separately for each particle $[m]$ using the same odometry measurement $\delta\mathbf{s}$. The complete motion model is described in Algorithm 3, where odometry measures the most recent pose difference $\delta\mathbf{s}$ and current heading θ.

Algorithm 3 Odometry Motion Model

```
$\delta_{rot1} = |\text{atan2}(\delta y, \delta x) - \theta|$
$\delta_{trans} = \sqrt{\delta x^2 + \delta y^2}$
$\delta_{rot2} = |\delta\theta - \delta_{rot1}|$
**for all** particles $[m] \in P$ **do**
    $d_{rot1}^{[m]} = \delta_{rot1} - \mathcal{N}(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})$
    $d_{trans}^{[m]} = \delta_{trans} - \mathcal{N}(\alpha_3 \delta_{trans} + \alpha_4 |\delta_{rot1} + \delta_{rot2}|)$
    $d_{rot2}^{[m]} = \delta_{rot2} - \mathcal{N}(\alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans})$
    $\delta x^{[m]} = d_{trans}^{[m]} \cos(d_{rot1}^{[m]})$
    $\delta y^{[m]} = d_{trans}^{[m]} \sin(d_{rot1}^{[m]})$
    $\delta\theta^{[m]} = \left| d_{rot1}^{[m]} + d_{rot2}^{[m]} + \theta^{[m]} \right|$
**end for**
```

The four α values exist to vary the size and shape of this distribution to model the anticipated odometry error. A researcher sets the α parameters to adjust this variance. A different vehicle often demands different α parameters resulting in a different distribution [61].

The distribution must be large enough to cover the actual measurement error and small enough to require few particles. In practice, this distribution is often bean-shaped to represent the error. This reflects the model being more confident in δ_{trans} and less in δ_{rot1} or δ_{rot2}.

Regardless of its shape, the odometry motion model distribution grows with each time t. As the vehicle travels a certain distance, the pose uncertainty grows and the distribution models this accordingly. Figure 5 displays the effect of this growing distribution on a simulated path that travels a fixed distance forward and makes two left turns.

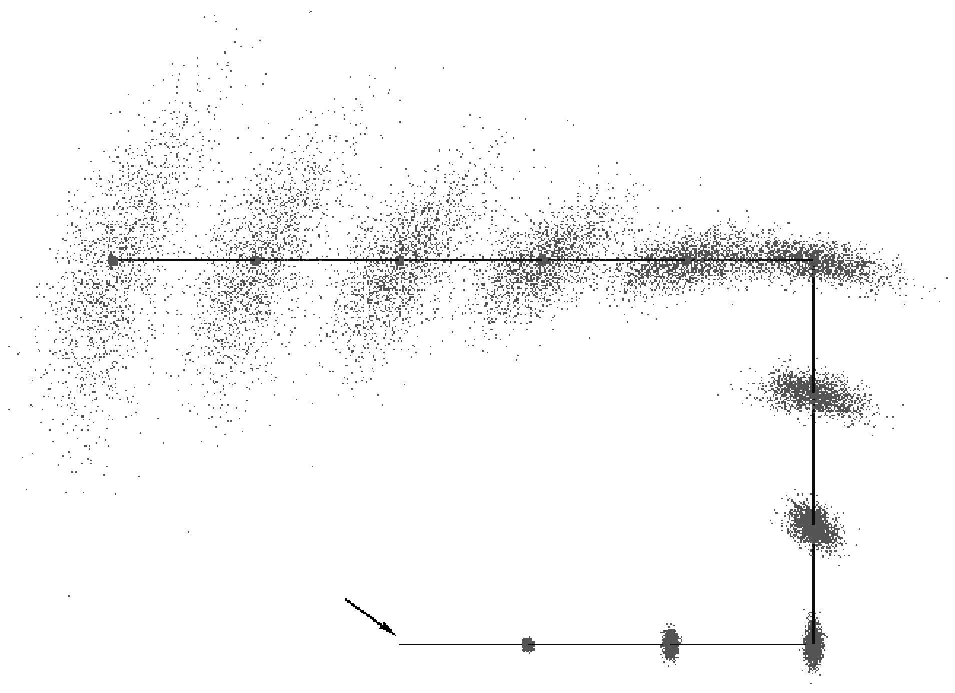

Figure 5. An example path showing a particle distribution spread using the odometry motion model. The vehicle begins at the arrow and travels along the path [61].

This motion model continues unchanged in this research, as described in the following chapter. Its α parameters allow the same process to create a plethora of different distributions depending on the specific odometry and environmental settings. This motion model does not describe the inertial or image inputs by themselves well, but remains effective in creating a desired distribution when combined with odometry.

2.4.2.2 Measurement Model.

The motion model only considers control input u_t to propagate the particles, and does not differentiate between accurate and inaccurate ones. To do this, FastSLAM needs a measurement model to incorporate observation z_t. Regardless of the specific details, the measurement model must stratify particle set P by incorporating current map knowledge, usually through z_t [61].

The measurement model distinguishes each particle $[m]$ by computing an error value for each one. FastSLAM uses this error to weight particle set P for resampling. Thus, the measurement model is the critical process to transition P from the prior to the posterior pose distribution [28]. This implementation uses a range scan model that observes each grid cell along each scan line. If a cell of map $\Theta^{[m]}$ is not consistent with current scan z_t, the model accumulates error for particle $[m]$ [33]. It builds up error in this manner for each $[m]$ at each time t to ensure the pose distribution of P maintains consistency with observation z_t.

Measurement models are less direct than the odometry motion model, reflecting the vast differences in observation sensors [61]. This research uses a range scan measurement model that builds error by occupancy grid cells [33]. However, a scanning technique that uses different sensor types, amounts, or placement must change its measurement model to factor this new physical arrangement. Another difference is in the map representation, where changing the occupancy grid also requires a different measurement model to observe map Θ. Details of the measurement model implemented in this research is presented in Chapter 3.

2.4.3 Alternative Map Representation.

While computational complexity between EKF based SLAM and FastSLAM is a common issue, the most significant contribution to running time is often associated with storing the map [20]. Especially when many particles each store their own copy, a large environment can require gigabytes of memory. Much research focuses on reducing this cost by making use of different map representation methods [6, 8, 24, 25].

An algorithm for dividing the global map into smaller regions allows those regions to be represented in a tree structure [25]. This reduces the number of landmarks to calculate at each step, and only consists of several more calculations to move between regions. The primary focus of this strategy is to close loops in the vehicle path. This means that returning to a previous location is successfully mapped to the same spot.

To prove the effect of this storage implementation, this strategy is used on an enormous data set of one million landmarks [24]. Such a data set is simply too large for an EKF to finish calculating, especially considering that memory was still the primary contributor in the improved implementation presented. The true map and constructed map of these tests are shown in Figure 6.

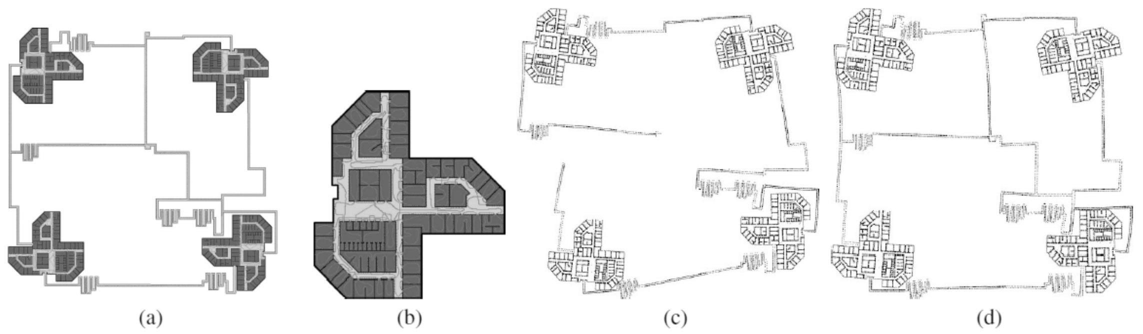

Figure 6. Maps from a million landmark data set. (a) is the true map of four buildings explored, (b) is a closeup of one building, (c) is the map estimate before closing the loop, and (d) is the final map estimate [24].

While the map generated from the algorithm in (d) does not completely replicate truth (a), it is able to improve itself from (c) by using landmark information. Even by visual inspection, (d) appears good enough to navigate with and succeeds in closing the loops to previous locations.

Most implementations store a single map type, but others maintain several local metric maps roughly the size of the sensor range. These are combined by maintaining an overall topological map and placing the local metric maps within it [8]. This results in less of a storage requirement and helps to keep local areas correctly placed relative to each other. Such a hybrid approach can also be used to separate the map into a hierarchy, updating one level at a time to produce one metric map at the end [6]. Figure 7 depicts the many steps involved in this process.

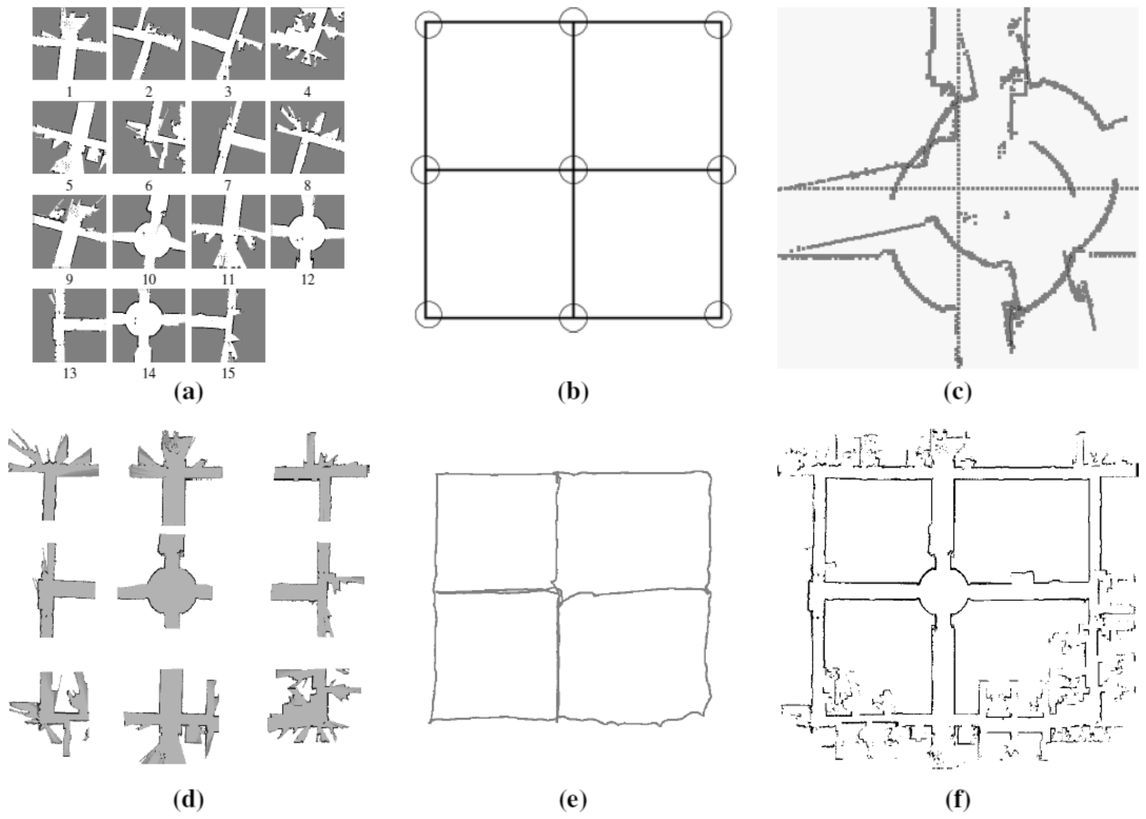

Figure 7. Maps built from a hybrid sequential process. The algorithm stores local metric maps (a) while building an overall topological map (b). It then finds offsets between poses (c) and places local maps into a global one (d). Finally, it corrects the vehicle path (e) to produce a final map (f) [6].

Such map representations are effective at computing large data sets and closing loops. Landmark storage is not the immediate focus of this work, which seeks to simplify the mapping process, not complicate it. This research holds both of these strategies as potential strategies to be applied later.

2.4.4 Comparing Results.

Due to the number of different outputs obtained from algorithms with different storage mechanisms, it is often difficult to compare them. Different algorithms use the same range sensors to interpret objects as occupied cells or separate landmarks. In the resulting maps, a metric occupancy grid does not resemble a topological map and both are different from a scatter plot of landmarks. Visual inspection of different representations can distinguish what looks like a believable map from one full of errors, but cannot distinguish between more similar results and is subject to human interpretation.

One paper presents a more precise method to compare the results of several different solutions [10]. This method considers vehicle path, one pose after another, to determine which output is closer to ground-truth. Not only is this a more quantitative measure than visual inspection, but it is more effective as well.

Because most algorithms first estimate each pose then build the map, comparing the poses of each algorithm results in a more significant difference. This work does not implement the specific method outlined, but instead uses a similar path comparison rather than simply discussing the differences in produced maps.

2.4.5 Advanced Techniques.

One significant factor that affects SLAM research significantly is the available hardware. Many solutions make use of a range finder to detect obstacles, either through RADAR, SONAR, or Light Detection and Ranging (LIDAR). While often expensive, these sensors are also incredibly accurate with LIDAR units maintaining millimeter to centimeter error resolution at distances up to tens of meters. A solution can detect distinct differences in adjacent readings to extract landmarks as done in early SLAM solutions. Alternatively, it can apply each scan to a metric grid map as done in this research.

The advantage of using such an accurate sensor comes from the small LIDAR error relative to the sensor. This is much different from the error of a pose estimate obtained through vehicle odometry or velocity measurements. Movement error from odometry or inertial measurements is not only much larger than the range errors but is also cumulative, often compounding quickly. In nearly all tests, the sensor ranges have much less error than vehicle measurements meaning the pose estimates cause map inconsistencies, not the sensor ranges.

To leverage this fact is the strategy of scan matching. By matching the proposal distribution to scan points, the solution incorporates range measurements into the pose estimate before the map is built. Scan matching is an adaptive technique that computes a more accurate proposal distribution by considering range inputs in the motion model to reduce pose uncertainty [28].

Presented scan matching research has improved path accuracy and reduced the pose distribution to require less particles [30]. This method succeeds in complicated environments using three dimensional scanners. The additional dimension generates a significant amount of data, but the principle applies to smaller and two dimensional environments as well [9].

The effect of scan matching is best illustrated by example situations. Around environment obstacles, situations arise where the algorithm incorporates the accurate ranges to narrow the pose distribution [28]. Less variance benefits the motion model by reducing pose error and allowing for fewer particles. Three map areas are pictured in Figure 8, each producing a different effect on the pose distribution.

By implementing scan matching improvements to a RBPF algorithm, researchers could reduce the number of particles required, cutting memory requirements and computation time [29]. Using the principle of scan matching, the most recent mea-

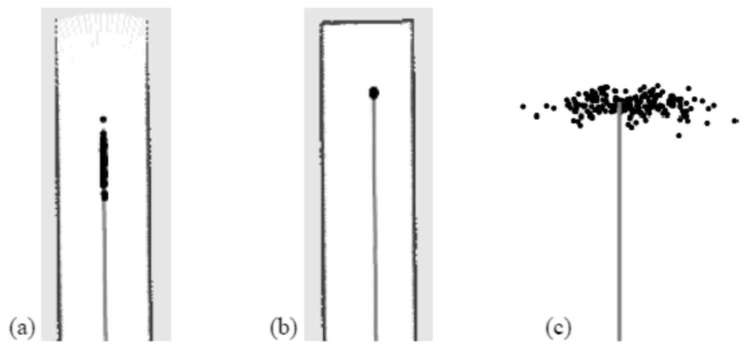

Figure 8. Mapping scenarios. In an open corridor (a), ranges detect walls on either side and shape the pose distribution. In a corner (b), more walls mean a more accurate estimate. Without walls (c), the motion model distribution is much larger [28].

surement is considered in addition to vehicle movement, resulting in a particle distribution with less variance. The RBPF then preceeds with weighting particle set P by comparing the current scan to each map.

As this work uses an accurate LIDAR, scan matching is certainly applicable. However, scan matching is used to decrease the particle distribution as an added improvement. It is not a necessary aspect of FastSLAM, so this research completes a functional SLAM solution before considering a scan matching application to the results.

Used in combination with scan matching, this research implements an adaptive resampling process to reduce the risk of removing accurate particles [29]. The resampling adapts by only resampling P once the weighting indicates a need, rather than at every time or after a fixed interval. This strategy is also implemented in this research, as both the RBPF and metric map techniques are similar.

Other algorithms correct the vehicle path separately from the map in an iterative process [50]. This improves the path of an otherwise incorrectly aligned map, but can only be calculated after the path is calculated, making it an offline solution. This work also dismisses this iterative strategy, as it focuses on an online implementation only.

There are many different SLAM algorithms, each with certain advantages in specific scenarios. The area of concern in this work is indoor, two dimensional environments. This simplifies the implementation of the algorithm and decreases complexity of the sensors. The result is less overall computation and memory requirement, allowing more tests to be conducted.

2.5 Machine Vision Localization & Mapping

Machine vision is a critical component of artificial intelligence and robotics research for same reason eyesight is a critical component of biological life. Image processing techniques can be applied immediately to accomplish navigation and positioning tasks just as humans and animals detect objects and movement through our vision. This section presents some recent research in tracking image features and using them for navigation and mapping.

This section discusses the tracking algorithms that provide a basis for FV-SIFT and cover its operation. It also presents recent research efforts in mapping using cameras, most of which is currently unavailable. The details of the image processing techniques in this work are discussed in the next chapter.

In this document, the word "feature" specifically refers to a spot of an image, whereas "landmark" is an important navigation point. An algorithm recognizes a physical object as a feature before interpreting it as a landmark in later steps.

2.5.1 Feature Tracking.

To save only relevant information, machine vision systems often use the concept of feature extraction, the process of finding points of interest in an image, called features [54]. Feature tracking finds the set of features F_A in one image and the set of features F_B in another image. It then matches a subset of $F_A \cap F_B$ to those features it finds between images. This process implies several difficult steps, such as recognizing what features were previously identified and tracking their movement between images.

Determining how well an algorithm recognizes features is difficult, as features can change in any combination of position, rotation, or size from one image to another [40]. In addition, features appear different under changing lighting conditions and from different perspectives.

The feature tracking problem is the first step in using vision for an application such as navigation. When computation hardware became available, researchers presented a process to determine which features to find and track [54]. The critical method is to select a limited number of features based on how well they will be tracked in subsequent images.

Subsequent methods in image feature extraction were shown to be more robust to change. One extraction method named the Scale-Invariant Feature Transform (SIFT) produces features that are more distinctive and less likely to be mistaken for others [40]. It focuses on the property of scale invariance, or how well the points being tracked are recognized as their scale changes. These good features are essential to the performance of the system, whose goal is to correctly label as many as possible in the next image. Although first shown to be effective for static object recognition, it has also been used for image navigation applications like SLAM [53].

SIFT produces accurate results, although subsequent research presents two successful alternatives, both aiming to reduce the required computation time when considering real-time applications. One method is Speeded-Up Robust Features (SURF) [5]. It works by approximating the features detection and description, decreasing the number of calculations and time required. Another method uses a different feature description by calculating differences in pixel bins, and is called Histogram of Oriented Gradients (HOG) [14]. Even though HOG is introduced for human detection, it achieves similar results as SURF for feature extraction and also requires much less time than SIFT.

The applicable navigation implementation was designed for SIFT to track features for use in navigation [65]. This work describes using a filter discussed in a later section to estimate a vehicle path. The algorithm has been used with both SIFT features and more recently faster SURF features, which provides the basis for the vision system used in this research. The implementation presented in the following chapter contains many of the same hardware components.

2.5.2 Image Feature Egomotion.

Many SLAM implementations have presented results using odometry and LIDAR, but others present the use of other input sensors. Some use vision cameras for hardware and algorithms to extract movement. The process of using sequential images for navigation is called egomotion, as used extensively for rover navigation when paired with odometry [48, 49]. Two of the Mars Exploration Rovers (MER) utilize this strategy, carrying stereo cameras to estimate an alternate path when traversing rough terrain where odometry is less accurate [41].

Perhaps the first published article using only vision features as input for the SLAM problem utilized SIFT and was presented at a similar time [53]. Alternative to filtering techniques, an iterative algorithm presents a reasonable method to generate a path [44]. Until recently, however, the application of these methods to SLAM has not been extensively explored.

A motivation for alternative inputs emerges as solutions become more accurate and computation speeds increase. This allows more experiments to be done using existing techniques with different sensors. For example, a RBPF SLAM solution using visual features as input can result in a more accurate map than pose estimates from odometry measurements [26, 55, 56]. These papers explain cost as a significant factor, as highly-accurate LIDAR units are often several thousands of dollars. While some can afford a limited number of such sensors, the application of SLAM capabilities often demands a cheaper solution.

Another consideration is the size and practicality of large sensor equipment. Money is not the only design cost, as LIDAR units often require more power than motors and computers and contribute significantly to weight. Machine vision may require more computation, but for a smaller application it also requires much less hardware. Cameras used have been as small as two lenses mounted onto a monitor, which produces clear enough images to create a map similar to odometry based poses [51, 22]. Other vehicles can be built with complicated rigs of several cameras taking a full panoramic view of the surroundings [35]. This allows for features to be tracked from one camera to another as the vehicle turns or revisits a location at a different orientation. This implementation results in a map with more obstacles defined, but requiring complicated software and significant memory to operate so many cameras at once.

The details of egomotion algorithms are often not provided in the presented research. The basic process relies on epipolar geometry to calculate focal points from stereo cameras [1]. The algorithms match image features between cameras and between time steps, then translate their movement into sensor motion. All egomotion calculations are discussed in the Methodology.

2.5.3 Additional Approaches & Applications.

Most research has focused on indoor or urban environments, seeking to create a floor plan or street layout. However, the SLAM problem applies outdoors just as easily with many of the same solutions. One stereo vision SLAM solution uses an EKF to map underwater [59]. The presented system is unique as it maps in three dimensions and calculates in real-time; most solutions ignore these otherwise applicable aspects.

A ground example presents a method of map storage specifically for environments where landmarks and obstacles have different heights [42]. This is not as applicable in hallways, where flat walls are restricted by a ceiling, but does significantly distinguish rocks from trees from buildings. Another difference in this research from others is that it uses a metric grid for map storage as many LIDAR solutions find it easy to do. This is in comparison to storing landmark locations as is otherwise done in vision solutions. This corresponds more closely to common RBPF solutions, implying their integration is possible.

A different approach beginning to emerge is that of machine learning. One could consider every SLAM solution to be an agent that learns the map with each subsequent measurement, but the presented solutions do not build upon this. Approaching SLAM from a machine learning perspective yields a solution that requires much less vision hardware and successfully navigates [32]. This implementation makes use of only a single camera, which observes the floor as opposed to walls. Its primary success was the determination that cameras are subject to blur effects not seen by other measurements, and mitigating this effect results in more accurate image data.

One approach using symmetric regions based on the HOG descriptor suggests that feature extraction methods are an effective option in further research [38]. Other recent research uses inertial data combined with egomotion to show that a calibrated system can perform in multiple environments with minimal drift [34]. In both cases, the results show that camera based methods are promising sensors for SLAM solutions where maintaining an accurate path and landmark locations are critical.

SLAM is still a relatively recent problem, and there is certainly more work that will be devoted to algorithms making use of different combinations of new sensor technologies and solution strategies. As more environments are explored and more applications are considered, additional algorithms work to provide them with functional solutions. It is likely that only few of the potential SLAM solutions have been presented, and there is a significant amount of robotic mapping yet to be done.

III. Methodology

This chapter presents a modified FastSLAM implementation that incorporates a novel combination of multiple input sensors. Where previous research used separate sensors to create separate paths, this work combines stereo image egomotion, integrated inertial measurements, and odometry to create a single path using a linear Kalman filter. This path combination will be referred to as the Multiple Integrated Navigation Sensors (MINS) system.

The MINS system sends its path to a FastSLAM implementation as its input path. FastSLAM uses this state to propagates its particles, then uses LIDAR ranges to measure error and generate occupancy grid maps. Lastly, it weights its particles based on the error and resamples them if necessary.

This chapter first begins with an overview of the novel MINS system as extended from principals in Chapter 1. The following sections cover the process of extracting motion from a set of stereo images, and a description of inertial data integration. Next is a description of how MINS filters pose estimates from egomotion, inertial movement, and odometry together. Following this, this chapter describes the process for integrating the combined poses into a RBPF implementation.

3.1 Overview

The inputs and outputs of a SLAM solution conform to a DBN on a fundamental level. Because this research stores the environment in a single map rather than a set of landmarks, it demands a different DBN representation than the one discussed in the first two chapters. Figure 9 depicts a DBN representing map Θ as a single hidden node rather than several landmarks. The single map condenses the desired solution into one object that affects each observation, instead of maintaining associations between every landmark at each time step t.

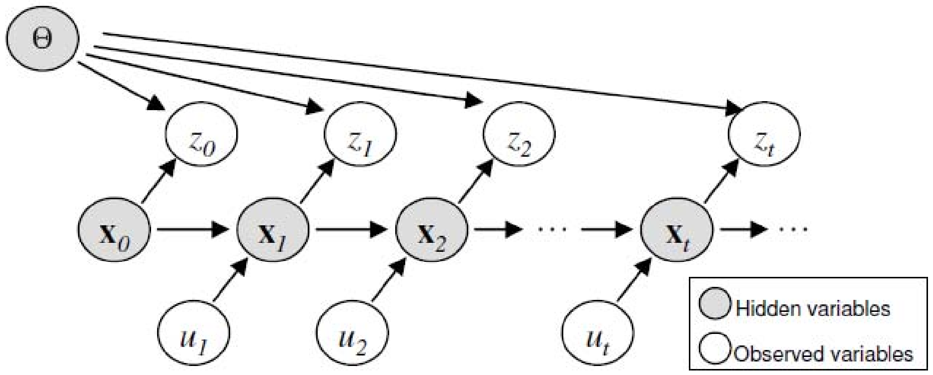

Figure 9. The SLAM problem as a dynamic Bayes network storing a single map. A vehicle travels through hidden states $\mathbf{x}_{0:t}$, observing controls $u_{1:t}$ and observations $z_{0:t}$. A single hidden map Θ affects all observations [8].

In this implementation with a single map, vehicle odometry acts as the controls $u_{1:t}$, driving the vehicle along its path. The vehicle takes LIDAR measurements as observations z_t of the environment Θ at each pose $\mathbf{s}_t$ within the state $\mathbf{x}_t$. Thus, a SLAM solution gives the values for the hidden nodes of the map Θ and current state $\mathbf{x}_t$.

When equipped with stereo cameras at equal height and inertial measurements in two dimensions, the vehicle gains additional input through image egomotion and numeric integration respectively, as discussed in detail later. The novel aspect of this research combines these inputs with odometry in a linear Kalman filter as the MINS system. MINS provides a navigation solution, leaving map construction to a separate FastSLAM implementation.

The linear Kalman filter takes a pose change measurement $\delta\mathbf{s}$ from each sensor as input. It uses these to maintain a state $\mathbf{x}$ and covariance $\mathbf{\Sigma}$ of a combined pose with associated accuracy. A separate particle filter then takes Kalman filter state $\mathbf{x}$ as its control input u to process a SLAM solution. The particle filter also incorporates the LIDAR scan z to construct maps, separating this functionality from the MINS sensor integration. An overall system diagram of MINS and FastSLAM is shown in Figure 10. It has four inputs from the four sensors, and maintains the particle filter states of a pose **s** and map Θ as system output.

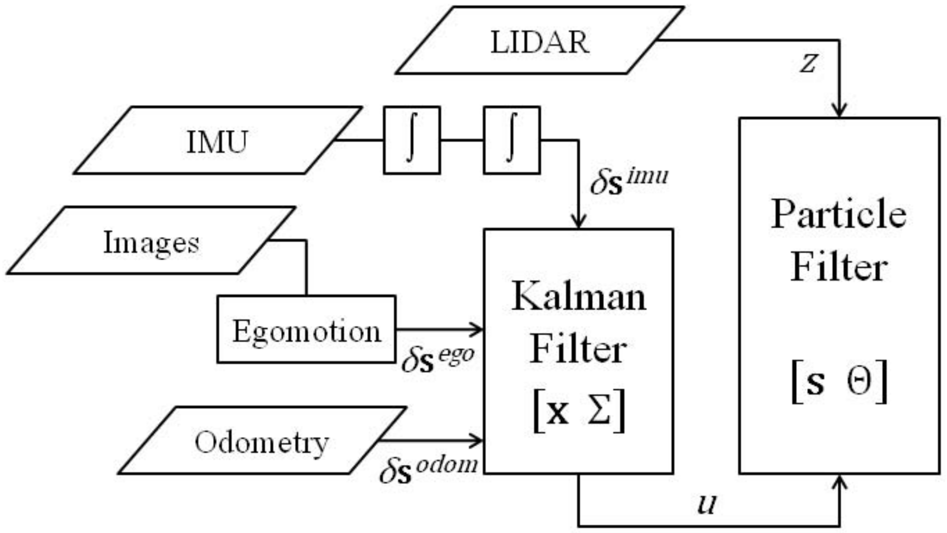

Figure 10. System diagram of the novel SLAM solution. A Kalman filter takes twice-integrated IMU measurements, egomotion from stereo images, and vehicle odometry as input. It then provides u to a particle filter that also takes LIDAR ranges z.

As the IMU records accelerations, MINS integrates each measurement to a velocity, then again to a position. These positions provide the Kalman filter with its prediction step. Each time an image pair is available, MINS calculates an egomotion pose and provides it to the Kalman filter as an observation step. Each odometry measurement is also a pose that provides another Kalman filter observation. Iterating over the series of prediction and observation steps updates the Kalman filter state **x** and covariance **Σ**.

After each odometry observation, at which time there may or may not have been an egomotion update, the Kalman filter sends its output to the particle filter as the control input u. Combined with LIDAR measurement z, FastSLAM maintains particle states each with a pose **s** and map Θ. These states are the hidden variables of the DBN, thus providing the SLAM solution.

3.2 Camera Input

This section first discusses FV-SIFT in more detail as it motivates this work. The hardware arrangement and data collection processes are identical for both systems, although this research only uses FV-SIFT as a comparison to previous methods. Following FV-SIFT, this section describes the process for extracting feature locations and egomotion algorithms. MINS calculates all image and inertial components independent of the FV-SIFT methods.

3.2.1 Image-Inertial Navigation.

This research begins with the FV-SIFT navigation system as presented in previous work [65]. Designed to use an EKF to estimate poses, a solution is also valid using an alternative UKF implementation [19]. The UKF version succeeds in navigating but does not reduce the position error of the original EKF solution and takes a longer time to compute. Regardless of its filtering technique, the system functions as depicted in Figure 11, resulting in a six Degrees of Freedom (DOF) pose change, with positions $\delta\mathbf{p}^n$ and rotations ψ in three dimensional space.

While an Inertial Measurement Unit (IMU) integrates its acceleration measurements to position data, two stereo cameras save images. An algorithm (either SIFT [40] or SURF [5]) extracts features from both images attempting to match and locate each feature. The system saves the best twelve features as landmarks to be matched in subsequent images. It continues to track matched landmarks and use their observed locations to update the pose from integrated IMU measurements.

FV-SIFT performs much better on aircraft than it does on a small indoor vehicle, but operates in either environment [65]. Additionally, it is highly subject to the accuracy of the IMU, resulting in a large error difference depending on the unit used. The path quality of the navigation system is also subject to feature loss, which results

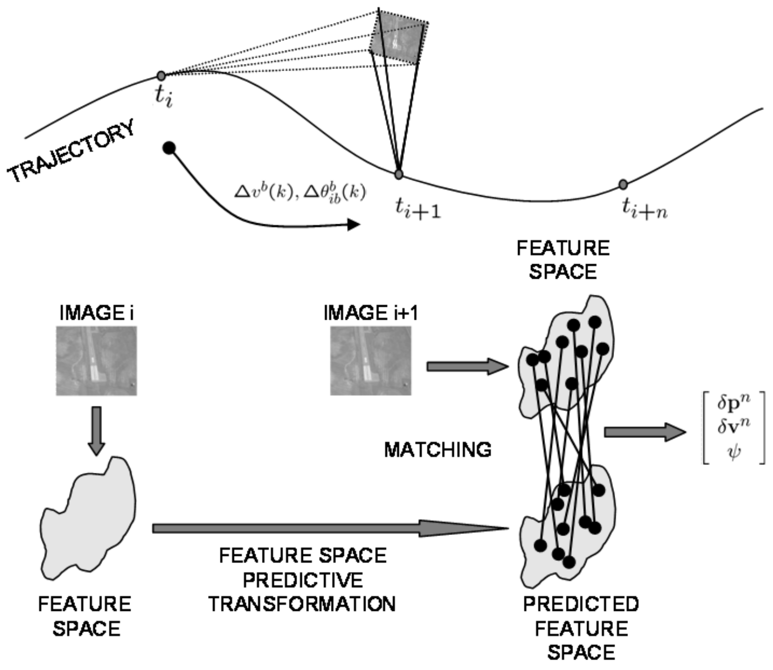
TRAJECTORY
t_i
t_{i+1}
t_{i+n}
$\Delta v^b(k), \Delta\theta^b_{ib}(k)$
FEATURE SPACE
IMAGE i
IMAGE i+1
MATCHING
$\begin{bmatrix} \delta\mathbf{p}^n \\ \delta\mathbf{v}^n \\ \psi \end{bmatrix}$
FEATURE SPACE
FEATURE SPACE PREDICTIVE TRANSFORMATION
PREDICTED FEATURE SPACE

Figure 11. Operation of the FV-SIFT system. It detects image features, predicts their future location, and attempts to match them in subsequent images with movement [65].

in unrecoverable errors after only a few seconds without observing some or all of its twelve landmarks. As a result of this loss, it relies completely on its IMU and rapidly drifts away.

Too much reliance on one system, either image features or inertial data, is motivation to seek more consistent navigation sensors. Sudden movements, collisions, and vibrations greatly affect an IMU but do not affect odometry as much. As this research concerns indoor ground vehicles, it assumes odometry is available as an input. A system with stereo cameras can adjust vehicle odometry and IMU measurements with its own path. Even if each of the inputs are not particularly accurate, filtering with an EKF or RBPF improves raw odometry using egomotion to calculate a path of its own [55].

The problems with FV-SIFT are the reason for the implementation decisions that follow in this chapter. It tightly couples the images and inertial data, but also tightly integrates itself to its EKF and cannot be easily extended due to specifics of its implementation. MINS separates the cameras from the IMU data until it combines them with odometry in a linear Kalman filter. This way, each input sensor does not rely on another and MINS is robust in areas that are problematic for FV-SIFT.

3.2.2 Feature Tracking & Stereopsis.

Camera egomotion is an older problem than SLAM, and researchers now seek to combine the separate fields [26, 55]. Several algorithms exist for calculating three dimensional motion can be applied to mobile robotics operating on a planar surface [1]. The mathematical solution using the epipolar geometry presented in this paper provides the basis for the feature processing used in this research. The implementation is based on stereopsis, the ability to approximate distances through stereo images.

To take advantage of stereopsis, MINS matches the SIFT features from the left camera image to those from the right camera image at the same time. The most straightforward way to compare two SIFT descriptors and determine a match between them is to take their Euclidean distance; however, the descriptor for each feature consists of 128 values. For computational purposes, MINS calculates the inverse cosine of the dot products of each pair of left camera descriptors $\mathbf{d}^L$ and right camera descriptors $\mathbf{d}^R$ to find a distance D between each descriptor pair. This is a similar measure to Euclidean distance but faster to calculate as shown in Equation 13.

$$D = \arccos(\mathbf{d}^L \cdot \mathbf{d}^R) \tag{13}$$

MINS then finds the minimum distance value d_j for each feature i, where feature index j the closest match to i in the other image. If d_j is less than ϱ times the second smallest distance value in D that is not d_j, MINS matches the feature and saves its pixel coordinates in P^L and P^R. The elements of each are assigned as described in Equation 14.

$$P^L,\ P^R = \left\{\mathbf{p}_i^L,\ \mathbf{p}_j^R : \min(D) < \varrho \min(D - d_j)\right\} \tag{14}$$

Because the vehicle travels on a flat surface with cameras at equal height, the stereo images should detect only horizontal differences and not vertical ones. Therefore, to reduce the number of false matches, MINS discards feature matches in the other image that measure greater than ϑ from horizontal. This adds an additional condition to the elements of P^L and P^R, which is described in Equation 15 where n_C is horizontal image resolution.

$$P^L,\ P^R = \left\{\mathbf{p}_i^L,\ \mathbf{p}_j^R : |\text{atan2}(y_j^R - y_i^L, x_j^R - x_i^L + n_C)| \leq \vartheta\right\} \tag{15}$$

At this point, MINS has a set of features correctly matched in the current image pair. The complete process for matching left and right features is shown in Algorithm 4.

Algorithm 4 Feature Matching

```
for all features $i$ at $\mathbf{p}_i^L$ in the left image do
    $D = \arccos(\mathbf{d}_i^L \cdot \mathbf{d}^R)$
    $j = \text{argmin}(D)$
    remove $d_j$ from $D$
    if $d_j < \varrho \min(D)$ then
        add $\mathbf{p}_i^L$ to $P^L$ and $\mathbf{p}_j^R$ to $P^R$
    end if
    $x^\Delta = x_j^R - x_i^L + n_C$
    $y^\Delta = y_j^R - y_i^L$
    if $|\text{atan2}(y^\Delta,\ x^\Delta)| > \vartheta$ then
        remove $\mathbf{p}_i^L$ from $P^L$ and $\mathbf{p}_j^R$ from $P^R$
    end if
end for
return $P^L$, $P^R$
```

The result of feature matching are the pixel coordinates P^L and P^R of the set of features to track in the left and right images, respectively. Before the coordinates can be accurately used, MINS removes lens distortion effects by performing twenty iterations through the CalTech distortion model procedure [65].

To do this, models for each camera contain calibrated parameters for these calculations. These include the principal point $\mathbf{p}_C^L$ and focal length $\mathbf{f}_C^L$ values, each with x and y components. Additional parameters include five values $\mathbf{k}_C^L$ indexed by brackets. Algorithm 5 uses this model to remove lens distortion from each $\mathbf{p}^L \in P^L$; the algorithm is identical for the right camera, substituting superscript R for each L.

Algorithm 5 Removing Lens Distortion

```
$\mathbf{h}_{distort} = (\mathbf{p}^L - \mathbf{p}_C^L)/\mathbf{f}_C^L$
$\mathbf{n} = \mathbf{h}_{distort}$
for 1 to 20 do
    $r = n_x n_x + n_y n_y$
    $k_{radial} = 1 + \mathbf{k}_C^L[1]r + \mathbf{k}_C^L[2]r^2 + \mathbf{k}_C^L[5]r^3$
    $h_y = 2\mathbf{k}_C^L[3]n_x n_y + \mathbf{k}_C^L[4](r + 2n_y^2)$
    $h_x = \mathbf{k}_C^L[3](r + 2n_x^2) + 2\mathbf{k}_C^L[4]n_x n_y$
    $\mathbf{n} = (\mathbf{h}_{distort} - \mathbf{h})/k_{radial}$
end for
return $\mathbf{p}^L = \mathbf{T}_C^L[-n_x \; n_y \; 1]$
```

After MINS executes Algorithm 5 to remove distortion effects from P^L and P^R, it can use their true coordinates as obtained through the last line. As measured, feature coordinates within camera pixels are unhelpful until projected into actual locations outside the cameras. To be useful for navigation, MINS uses epipolar geometry to measure the physical location of each feature [1].

Epipolar geometry calculations require information from models of both cameras, which have saved images at $m_C \times n_C$ pixel resolution. The first step in determining the physical location is to transform the coordinates of the features. This is done using a Direction Cosine Matrix (DCM) to convert pixels to locations $\mathbf{T}_C^L$ and another to convert locations to the correct reference frame $\mathbf{C}_C^L$. Equation 16 expresses a feature in coordinates appropriate to the left camera. MINS performs the same calculations for each feature using the models for the left and right camera.

$$\mathbf{o}^L = \mathbf{C}_C^L \left(\mathbf{T}_C^L\right)^{-1} \begin{bmatrix} x_{\mathbf{p}}^L - \lceil m_C/2 \rceil \\ y_{\mathbf{p}}^L - \lceil n_C/2 \rceil \\ 1 \end{bmatrix}^T \tag{16}$$

The algorithm then combines transformed coordinates $\mathbf{o}^L$ and $\mathbf{o}^R$ to take advantage of stereopsis by using the distance from the right to the left camera $\mathbf{o}_C$. MINS computes this using multiple inner products in Equation 17 to obtain the location $\mathbf{f}^L$ as seen from the left side.

$$\mathbf{f}^L = \frac{(\mathbf{o}^L \cdot \mathbf{o}^R)(\mathbf{o}^R \cdot \mathbf{o}_C) - (\mathbf{o}^R \cdot \mathbf{o}^R)(\mathbf{o}^L \cdot \mathbf{o}_C)}{(\mathbf{o}^L \cdot \mathbf{o}^L)(\mathbf{o}^R \cdot \mathbf{o}^R) - (\mathbf{o}^L \cdot \mathbf{o}^R)^2} \tag{17}$$

A similar calculation in Equation 18 gives the location $\mathbf{f}^R$ of one feature with respect to the right side.

$$\mathbf{f}^R = \frac{(\mathbf{o}^L \cdot \mathbf{o}^L)(\mathbf{o}^R \cdot \mathbf{o}_C) - (\mathbf{o}^L \cdot \mathbf{o}^R)(\mathbf{o}^L \cdot \mathbf{o}_C)}{(\mathbf{o}^L \cdot \mathbf{o}^L)(\mathbf{o}^R \cdot \mathbf{o}^R) - (\mathbf{o}^L \cdot \mathbf{o}^R)^2} \tag{18}$$

With both $\mathbf{f}^L$ and $\mathbf{f}^R$, MINS simply averages the two values to obtain feature location $\mathbf{l}$. Each $\mathbf{l}$ is expressed relative to the vehicle cameras and thus relative to the current vehicle pose. This allows MINS to maintain a record of all image features seen in both images with their observed locations.

The overall process for obtaining each location $\mathbf{l}$ from P^L and P^R is described in Algorithm 6. Vectors are denoted by bold lowercase letters and matrices by bold uppercase letters; multiplication and dot products are written explicitly.

Each location $\mathbf{l}$ must be rotated from its value in Algorithm 6 to match the coordinate system of the vehicle. With all two dimensional tracked feature locations $\mathbf{l}$, MINS considers the these feature locations landmarks. It then uses them to construct an estimate of vehicle motion independent of other common inputs such as inertial

Algorithm 6 Feature Locations from Pixel Coordinates

```
$\mathbf{o}^L = \mathbf{C}_C^L(\mathbf{T}_C^L)^{-1}\begin{bmatrix} x_\mathbf{p}^L - \lceil m_C/2 \rceil & y_\mathbf{p}^L - \lceil n_C/2 \rceil & 1 \end{bmatrix}$
$\mathbf{o}^R = \mathbf{C}_C^R(\mathbf{T}_C^R)^{-1}\begin{bmatrix} x_\mathbf{p}^R - \lceil m_C/2 \rceil & y_\mathbf{p}^R - \lceil n_C/2 \rceil & 1 \end{bmatrix}$
$\mathbf{a} = \mathbf{o}^L \cdot \mathbf{o}^L$
$\mathbf{b} = \mathbf{o}^L \cdot \mathbf{o}^R$
$\mathbf{c} = \mathbf{o}^R \cdot \mathbf{o}^R$
$\mathbf{d} = \mathbf{o}^L \cdot \mathbf{o}_C$
$\mathbf{e} = \mathbf{o}^R \cdot \mathbf{o}_C$
$\mathbf{f}^L = (\mathbf{b}\ \mathbf{e} - \mathbf{c}\ \mathbf{d})/(\mathbf{a}\ \mathbf{c} - \mathbf{b}^2)$
$\mathbf{f}^R = (\mathbf{a}\ \mathbf{e} - \mathbf{b}\ \mathbf{d})/(\mathbf{a}\ \mathbf{c} - \mathbf{b}^2)$
**return** $\mathbf{l} = (\mathbf{f}^L\mathbf{o}^L + \mathbf{f}^R\mathbf{o}^R)/2$
```

or odometry measurements.

3.2.3 Stereo Egomotion.

MINS uses each landmark location $\mathbf{l}_t^i$, where i is the landmark index and t is the current time step. It consists of rectangular position x_t^i and y_t^i as measured from the current pose of the vehicle, as defined in Equation 19.

$$\mathbf{l}_t^i \equiv \begin{bmatrix} x_t^i \\ \\ y_t^i \end{bmatrix} \tag{19}$$

The algorithm assumes that the only thing moving in the environment is the vehicle and not the landmarks. Therefore, it measures motion from landmark i tracked at time t as it has a current location $\mathbf{l}_t^i$ and a previous location $\mathbf{l}_{t-1}^i$. This makes it possible to extract a vehicle pose change, because the algorithm interprets motion in each landmark to be a result of vehicle motion.

There are K tracked landmarks, which changes at each time t. MINS attempts to reduce the number of bad matches by eliminating those that travel more than σ_l standard deviations from the mean distance $\bar{d}$. The calculation in Equation 20 removes outliers from corrupting observed motion, where σ_d is the statistical standard deviation of the numerator for each landmark i.

$$d^i = \frac{\left||x^i_t - x^i_{t-1}| + |y^i_t - y^i_{t-1}| - \bar{d}\right|}{(\sigma_d)^2} \tag{20}$$

MINS then uses a polar representation to measure movement. The rectangular coordinates of $\mathbf{l}^i$ are an angle and a radius calculated in Equation 21.

$$\begin{bmatrix} \theta^i \\ r^i \end{bmatrix} \equiv \begin{bmatrix} \text{atan2}(y^i, x^i) \\ \sqrt{(x^i)^2 + (y^i)^2} \end{bmatrix} \tag{21}$$

Finally, MINS measures the vehicle change from $t-1$ to t by subtracting $\mathbf{l}^i_t$ from $\mathbf{l}^i_{t-1}$ in polar form. Note that this is the negative of typical motion, as the algorithm is translating landmark movement in one direction as vehicle movement in the other. This subtraction results in a position difference for each landmark i is Equation 22.

$$\begin{bmatrix} \delta\theta^i \\ \delta r^i \end{bmatrix} = \begin{bmatrix} \theta^i_{t-1} \\ r^i_{t-1} \end{bmatrix} - \begin{bmatrix} \theta^i_t \\ r^i_t \end{bmatrix} \tag{22}$$

As a final step, the algorithm takes the mean of each landmark difference represented by elements $\bar{\theta}$ and $\bar{r}$, then converts the coordinates back to rectangular form. This creates egomotion pose difference $\delta\mathbf{s}^{ego}_t$, computed in Equation 23.

$$\delta\mathbf{s}^{ego}_t = \begin{bmatrix} \bar{r}\cos(\bar{\theta}) \\ \bar{r}\sin(\bar{\theta}) \\ \bar{\theta} \end{bmatrix} \tag{23}$$

The overall process for extracting pose change $\delta\mathbf{s}^{ego}_t$ from the set L_t of all locations $\mathbf{l}$ at time t is described in Algorithm 7. A bar over a variable indicates its mean and σ indicates a statistical standard deviation.

As MINS tracks K landmarks, Algorithm 7 calculates the angle θ^i and distance r^i of each landmark i for both $t-1$ and t. It then calculates each difference d^i between

Algorithm 7 Landmark Egomotion

```
for all time steps $t$ do
    for $i = 1$ to $K$ do
        $n = |x_t^i - x_{t-1}^i| + |y_t^i - y_{t-1}^i|$
        $d^i = |n - \bar{d}|/(\sigma_n)^2$
        if $d^i > \sigma_d$ then
            remove $\mathbf{l}^i$ from $L_t$ and from $L_{t-1}$ // this decreases $K$
        end if
    end for
    for $i = 1$ to $K$ do
        $\theta_t^i =$atan2$(y_t^i, x_t^i)$
        $r_t^i = \sqrt{(x_t^i)^2 + (y_t^i)^2}$
        $\theta_{t-1}^i =$atan2$(y_{t-1}^i, x_{t-1}^i)$
        $r_{t-1}^i = \sqrt{(x_{t-1}^i)^2 + (y_{t-1}^i)^2}$
    end for
    $\delta\theta = \theta_{t-1} - \theta_t$
    $\delta r = r_{t-1} - r_t$
    $\delta\mathbf{s}_t^{ego} = [\bar{r}\cos(\bar{\theta})\ \bar{r}\sin(\bar{\theta})\ \bar{\theta}]^T$
end for
```

the landmark locations in the most recent interval. MINS calculates means $\bar{r}$ and $\bar{\theta}$ as the total movement of all landmarks.

Assuming the landmarks are stationary, MINS interprets their movement relative to the vehicle as vehicle movement. It converts $\bar{r}$ and $\bar{\theta}$ back to rectangular coordinates, which is the difference in vehicle pose. This difference $\delta\mathbf{s}_t^{ego}$ serves as the pose difference between $t-1$ and t as observed by egomotion.

3.3 Inertial Input

Simply calculating $\delta\mathbf{s}_t^{ego}$ is not a viable navigation path by itself. As mentioned earlier in this chapter, a navigation solution takes a risk when relying on a single input due to specific disadvantages in the sensor and the errors that arise from them. The reason for the success of the image aided inertial navigation system discussed is indeed the combination of its two inputs [65].

The MINS system presented in this research also uses an IMU to obtain a similar inertial input pose difference $\delta\mathbf{s}_t^{imu}$. With comparable estimate from the cameras in $\delta\mathbf{s}_t^{ego}$, it seeks a similar estimate from the IMU in $\delta\mathbf{s}_t^{imu}$, and as measured by odometry in $\delta\mathbf{s}_t^{odom}$. Vehicle odometry records data directly, but MINS must calculate $\delta\mathbf{s}_t^{imu}$ from raw IMU measurements before combining the pose differences.

The IMU contains an accelerometer and a gyroscope that measure specific forces and rotational velocities [63]. With these instruments, the IMU records accelerations due to gravity and external forces that cause movement. In three dimensions, it records measurements in six DOF. This research only concerns movement in two dimensions, or three DOF, with a stationary IMU. Because this research limits the area of interest to flat surfaces, it assumes zero vertical movement, roll, and pitch.

These other values are ignored and MINS is left with three orthogonal forces, one of which is due to gravity and is discarded. MINS only considers the other two accelerations $\ddot{x}$, $\ddot{y}$, and yaw velocity ω. Therefore, the IMU measures two dimensional instantaneous acceleration $\mathbf{a}_t$ at time t in Equation 24.

$$\mathbf{a}_t \equiv \begin{bmatrix} \ddot{x}_t \\ \ddot{y}_t \end{bmatrix} \tag{24}$$

Because navigating requires a position, IMU measurements $\mathbf{a}_t$ must be integrated twice, and ω_t must be integrated once between each time interval from $t-1$ to t [63]. To limit the effect of drifting due to this integration, programs interfacing with an IMU commonly track a bias [65]. The vehicle in this research always begins stationary, so MINS leverages information from the available initial data to correct this.

Until the odometry indicates any vehicle movement, MINS calculates IMU measurement bias b_t by taking a moving average of ω_t. This is shown in Equation 25, where τ is the total number of measurements included in b_{t-1} [65].

$$b_t = \frac{\omega_t + \tau b_{t-1}}{\tau + 1} \tag{25}$$

With b_t calculated, MINS tracks dt, the time interval since previous IMU measurement $\mathbf{a}_{t-1}$. It uses each of these values to approximates a three DOF instantaneous velocity $\mathbf{v}_t$ in Equation 26 [63].

$$\mathbf{v}_t \equiv \begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{\theta}_t \end{bmatrix} = \begin{bmatrix} \dot{x}_{t-1} + dt\ \ddot{x}_t \\ \dot{y}_{t-1} + dt\ \ddot{y}_t \\ \omega_t - b_t \end{bmatrix} \tag{26}$$

To mitigate further acceleration errors from producing unrealistic velocities, MINS includes a novel step. Because it knows the physical limits of the vehicle, it limits its speed v_t to the known vehicle maximum v_{max}. By computing speed as the magnitude of linear velocity, MINS scales the elements of $\mathbf{v}_t$ by integrated overall speed v_t at time t without affecting the angle of travel. This calculation, shown in Equation 27, is only applied if v_t is greater than v_{max}.

$$\mathbf{v}_t = \begin{bmatrix} \dot{x}_t / \sqrt{(\dot{x}_t)^2 + (\dot{x}_t)^2} \\ \dot{y}_t / \sqrt{(\dot{x}_t)^2 + (\dot{y}_t)^2} \\ \dot{\theta}_t \end{bmatrix} \tag{27}$$

A similar process to the velocity calculation performs numeric integration to bring velocity to position [63]. Equation 28 approximates the current IMU estimate of the pose difference $\delta \mathbf{s}_t^{imu}$.

$$\delta \mathbf{s}_t^{imu} = \frac{dt}{2}\Big(\mathbf{v}_{t-1} + \mathbf{v}_t\Big) \tag{28}$$

MINS handles tracking b_t and limiting $\mathbf{v}_t$ for each time step t of IMU data at time interval dt. MINS must store only the current bias b_t, the number of time steps τ, the previous velocity $\mathbf{v}_{t-1}$, and nothing more. The full process for obtaining $\delta \mathbf{s}_t^{imu}$

from $\mathbf{a}_t$ is shown in Algorithm 8.

Algorithm 8 IMU Integration

```
if no odometry movement then
    $b_t = (\omega_t + \tau b_{t-1})/(\tau + 1)$
    increment $\tau$
end if
$\mathbf{v}_t = \mathbf{v}_{t-1} + dt\ \mathbf{a}_t$
$\dot{\theta} = \omega_t - b_t$
$v_t = \sqrt{(\dot{x}_t)^2 + (\dot{y}_t)^2}$
if $v_t > v_{max}$ then
    $\dot{x}_t = \dot{x}_t / v_t$
    $\dot{y}_t = \dot{y}_t / v_t$
end if
$\delta\mathbf{s}_t^{imu} = dt(\mathbf{v}_t + \mathbf{v}_{t-1})/2$
```

Even with the corrections applied, the most noticeable issue with integrating twice is that minor errors in $\mathbf{a}_t$ cause $\delta\mathbf{s}_t^{imu}$ to grow with each measurement, and the path obtained using this difference drifts more rapidly with each measurement.

3.4 Control Input Kalman Filter

The MINS system combines three separately obtained pose differences to determine the final control input $\delta\mathbf{s}_t$ at each time t. It combines $\delta\mathbf{s}_t^{imu}$ from the IMU, $\delta\mathbf{s}_t^{ego}$ from the cameras, and $\delta\mathbf{s}_t^{odom}$ from odometry in a linear Kalman filter. In the same manner as previously demonstrated, a Kalman filter allows the three inputs to produce a combined pose difference $\delta\mathbf{s}_t$. Each of these inputs have different uncertainties that vary between measurements, as changing environments affect each sensor differently.

Kalman filtering is advantageous to a simpler combination because MINS can update the filter with different inputs at different time steps. This is critical because each sensor measures at its own interval and is not always constant. A Kalman filter continues to approximate the pose with any combination of the sensors available.

This means the filter continues to estimate given only one input, even if both of the other inputs are unavailable or untrusted. The filter carries only its current pose $\mathbf{s}_t$ as its state $\mathbf{x}_t$, with covariance $\mathbf{\Sigma}_t$.

Kalman filters are often described as a series of predictions and observations that maintain $\mathbf{x}_t$ and $\mathbf{\Sigma}_t$ [43]. The IMU collects data fast, so MINS offers $\delta\mathbf{s}_t^{imu}$ to the linear Kalman filter as the prediction. After finishing each inertial integration, the filter prediction updates the state prior $\hat{\mathbf{x}}_t$ in Equation 29. Here, 3×3 matrix $\mathbf{F}$ directly relates the prediction $\delta\mathbf{s}_t^{imu}$ to state variables $\hat{\mathbf{x}}_t$ as an identity.

$$\hat{\mathbf{x}}_t = \mathbf{F}\big(\mathbf{x}_{t-1} + \delta\mathbf{s}_t^{imu}\big), \text{ where } \mathbf{F} = \mathbf{I}_3 \tag{29}$$

To obtain the covariance prior $\hat{\mathbf{\Sigma}}_t$, the Kalman filter considers the variances of the inertial measurements σ_{acc} and σ_{gyro} along the diagonal of $\mathbf{Q}$. The Kalman filter propagates these variances to $\hat{\mathbf{\Sigma}}_t$ in Equation 30, where $\mathbf{G}$ relates the variances in $\mathbf{Q}$ to the values in prior covariance matrix $\hat{\mathbf{\Sigma}}_t$.

$$\hat{\mathbf{\Sigma}}_t = \mathbf{G}\mathbf{Q}\mathbf{G}^T = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_{acc} & 0 \\ 0 & \sigma_{gyro} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sigma_{acc} & \sigma_{acc} & 0 \\ \sigma_{acc} & \sigma_{acc} & 0 \\ 0 & 0 & \sigma_{gyro} \end{bmatrix} \tag{30}$$

Prior state $\hat{\mathbf{x}}_t$ produces a path that considers only measurements from the IMU. When odometry or egomotion inputs are available, MINS provides the Kalman filter with $\delta\mathbf{s}_t^{odom}$ and $\delta\mathbf{s}_t^{ego}$ as observations. Each observation brings the prior state and covariance to their posterior values. The Kalman filter performs the same process for odometry and image updates, but maintains a different error model $\mathbf{R}_t$ for the different inputs at time t.

For odometry, the Kalman filter scales the measured pose difference by the time

δt since the last odometry update. Equation 31 displays the observation variance of odometry $\mathbf{R}_t^{odom}$ with a constant heading variance of 2π radians.

$$\mathbf{R}_t^{odom} = \begin{bmatrix} \delta t \left|\delta x^{odom}\right| & 0 & 0 \\ 0 & \delta t \left|\delta y^{odom}\right| & 0 \\ 0 & 0 & 2\pi \end{bmatrix} \tag{31}$$

For image updates, the Kalman filter uses the number of landmarks k tracked in the current image pair as a basis the variance, both position coordinates and heading. Equation 32 displays the observation variance of an egomotion update $\mathbf{R}_t^{ego}$.

$$\mathbf{R}_t^{ego} = \begin{bmatrix} 10/k & 0 & 0 \\ 0 & 10/k & 0 \\ 0 & 0 & 100\pi/k \end{bmatrix} \tag{32}$$

The Kalman filter uses $\delta\mathbf{s}_t$ and $\mathbf{R}_t$ for the appropriate sensor to make an observation of current state $\mathbf{x}_t$. Just as $\mathbf{F}$ relates the prediction directly to the state, 3×3 matrix $\mathbf{H}$ relates the observation to the state and is also an identity. Equation 33 shows the Kalman gain $\mathbf{K}_t$ and posterior covariance $\mathbf{\Sigma}_t$ obtained from a function call to a U-D decomposition method [7]. This function exists within the filtering library.

$$\left\langle \mathbf{K}_t,\ \mathbf{\Sigma}_t \right\rangle = \text{ObserveUD}\left(\hat{\mathbf{\Sigma}}_t,\ \mathbf{H} = \mathbf{I}_3,\ \mathbf{R}_t\right) \tag{33}$$

To observe its state, the Kalman filter multiplies each element of $\mathbf{K}_t$ by each element of a measurement residual. The residual is the pose difference between the prior state and the observed pose. Equation 34 describes the calculation for obtaining the posterior state $\mathbf{x}_t$ from the prior state $\hat{\mathbf{x}}_\mathbf{t}$.

$$\mathbf{x}_t = \hat{\mathbf{x}}_t + \mathbf{K}_t\left(\delta\mathbf{s}_t - \hat{\mathbf{x}}_t\right) \tag{34}$$

With the observation equations completed, the Kalman filter stores a posterior state $\mathbf{x}_t$ and covariance $\mathbf{\Sigma}_t$ for time t. As MINS repeatedly updates its filter with desired output $\mathbf{x}_t$ always available, it becomes the total pose difference $\delta\mathbf{s}_t$. In this way, the linear Kalman filter generates a localization path for mapping. The path created by each $\delta\mathbf{s}_t$ is considered the output of the MINS system.

3.5 FastSLAM Implementation

MINS provides its pose difference $\delta\mathbf{s}_t$ to the particle filter as control input u_t with an associated 3×3 covariance $\mathbf{\Sigma}_t$. With $\mathbf{x}_t \equiv \delta\mathbf{s}_t \equiv u_t$, the RBPF has a two dimensional path to use for its FastSLAM implementation.

3.5.1 Motion Model.

The first FastSLAM step dictates the particles be propagated according to the control input u_t from the MINS path. This control input corresponds to pose change $\delta\mathbf{s}_t$ in the odometry motion model from previous work [61]. By using the Kalman filter, FastSLAM gains covariance $\mathbf{\Sigma}_t$, containing values not available when using a single sensor. Since the goal of particle propagation is to model the pose uncertainty, FastSLAM incorporates covariance matrix $\mathbf{\Sigma}_t$ into the odometry motion model.

Due to the error models of the Kalman filter inputs, each $\mathbf{\Sigma}$ is block-diagonal in addition to being symmetric. It separates the coordinates from the heading standard deviation, taking the values shown in Equation 35.

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & 0 \\ \sigma_{xy} & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{35}$$

The least intrusive manner to incorporate the values in $\mathbf{\Sigma}$ is to modify the motion model parameters. This does not affect the odometry motion model operation, which remains useful as it continues to accurately describe wheeled vehicle movement. The uncertainty in rotation by translation α_2 and translation by rotation α_4 model expected vehicle error, but correspond with the zeros of $\mathbf{\Sigma}$. Because of this, FastSLAM does not modify these but uses configured constants instead.

Uncertainty in translation by translation α_3 and rotation by rotation α_1 correspond respectively with the upper and lower blocks of $\mathbf{\Sigma}$. Therefore, FastSLAM obtains its four motion model parameters in Equation 36 by modifying new parameters, a_1 through a_4.

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} \left(\sigma_\theta^2\right) a_1 \\ a_2 \\ \left(\sigma_x^2 + \sigma_y^2 + \sigma_{xy}\right) a_3 \\ a_4 \end{bmatrix} \tag{36}$$

It is worth noting that the units are not consistent using this method. The elements of $\mathbf{\Sigma}$ have units of distance and heading, corresponding to those of its state, which is a pose. The motion model uses each of its α parameters as uncertainty in pose change, which has units of change in distance and change in heading. The specific elements of $\mathbf{\Sigma}$ could be subtracted between times t and $t-1$ to obtain comparable values, but FastSLAM uses Equations 35 and 36 due to implementation details.

FastSLAM operates the motion model for M particles as described in the previous chapter using each of these α parameters. The difference is that configured parameters a_1 and a_3 are modified by covariance $\mathbf{\Sigma}$. After each particle $[m]$ propagates its current pose $\mathbf{s}_t^{[m]}$, the FastSLAM algorithm then builds the corresponding map $\Theta^{[m]}$ and weights particle $[m]$ within particle set P.

3.5.2 Measurement Model.

After the motion model discussed in the last chapter propagates each particle, the RBPF must rank particle set P using the available sensor data combined with the current state. Thus, the RBPF applies a sensor measurement model to calculate an error for each particle [33]. A more correct pose propagation should reflect a better match to the current range scan when compared to the map stored by that particle. FastSLAM then uses the error accumulated through the current time t to resample the particle distribution.

The measurement model is a significant contributor to the FastSLAM running time, requiring one iteration for each range scan for each particle $[m]$. This is more of a factor than the motion model or resampling operations, which do not consider the LIDAR range scans.

In this implementation, the sensor data consists of LIDAR scans and each particle maintains a current state of the map. A different sensor or map representation technique requires a different algorithm that provides the same measurement functionality. As an intuitive approach, each particle in this implementation stores the map as an occupancy grid, a large matrix with each cell of size g_{res} containing a probability of that area being occupied by an object.

The measurement model compares the current LIDAR scan to each particle map, measuring consistency with the environment. Observing the environment is the necessary function of the measurement model, but its accuracy is limited by the map resolution g_{res}.

Each LIDAR scan consists of a range r_s at an angle θ_s, where each particle has a pose $\mathbf{s}_t^{[m]}$ and map $\Theta^{[m]}$ used in this model. If the current range r_s is within valid measurements r_{min} to r_{max}, the measurement model calculations begin with Equation 37 by determining the current scanned position $\mathbf{q}_t$ by pose addition.

$$\mathbf{q}_t = \mathbf{s}_t^{[m]} + \begin{bmatrix} r_s \cos(\theta_s) \\ r_s \sin(\theta_s) \\ 0 \end{bmatrix} \tag{37}$$

The model then makes similar calculations to those used for applying the current ranges to the map. It begins by calculating the ratio ρ of range scan r_s to the occupancy grid resolution g_{res} in Equation 38. The measurement model uses ρ as a step count as it traces each scan the same way each map uses ρ in to build its grid.

$$\rho = \left\lceil \frac{r_s}{g_{res}} \right\rceil \tag{38}$$

The measurement model uses ρ to calculate position difference coordinates $\mathbf{d}$ in Equation 39. To do this, it scales the difference between $\mathbf{s}_t^{[m]}$ and $\mathbf{q}_t$ by

$$\mathbf{d} \equiv \begin{bmatrix} x^d \\ y^d \end{bmatrix} = \frac{1}{\rho} \begin{bmatrix} x^q - x_t^{[m]} \\ y^q - y_t^{[m]} \end{bmatrix} \tag{39}$$

Because the map uses difference $\mathbf{d}$ as an interval to apply scans, the measurement model does the same calculations to observe the map. To step from $\mathbf{s}_t^{[m]}$ to $\mathbf{q}_t$ along the grid, it steps along the scanned line using coordinates $\mathbf{p}$ in Equation 40. FastSLAM builds the map locations along the scan line from 0 to $\rho + 1$ using index j, so the model uses these positions to view the same grid coordinates.

$$\mathbf{p} \equiv \begin{bmatrix} x^p \\ y^p \end{bmatrix} = \begin{bmatrix} x_t^{[m]} + (j)x^d \\ y_t^{[m]} + (j)y^d \end{bmatrix} \tag{40}$$

To incorporate the map, the model peeks at the current particle occupancy grid value $\Theta^{[m]}$ at each location $\mathbf{p}$. If the cell has been previously observed as occupied when the current scan is not at its end, the measurement model updates map error

$e_t^{[m]}$ for particle $[m]$ in Equation 41. Here, the model scales $e_t^{[m]}$ by occupancy range g_{pr} and calculates an approximate distance from index j.

$$e_t^{[m]} = e_t^{[m]} + \Theta[x^p][y^p]^{[m]} \left(\frac{(\rho + 1 - j) g_{res}}{g_{pr}} \right) \tag{41}$$

When j reaches $\rho + 1$, the model expects location $\mathbf{p}$ to be occupied. If this cell has been previously observed as unoccupied, it updates the map error $e_t^{[m]}$ for particle $[m]$ using Equation 42. The model scales $e_t^{[m]}$ by occupancy range g_{pr} and uses grid resolution g_{res} for a minimum distance.

$$e_t^{[m]} = e_t^{[m]} - \Theta[x^p][y^p]^{[m]} \left(\frac{g_{res}}{g_{pr}} \right) \tag{42}$$

To keep error values $e_t^{[m]}$ low and factor recent observations more than earlier ones, the measurement model decays the previous error at each time [52]. It does this by multiplying constant discount factor γ to the previous error $e_{t-1}^{[m]}$ before adding current error $e_t^{[m]}$ in Equation 43.

$$e_t^{[m]} = \gamma e_{t-1}^{[m]} + e_t^{[m]} \tag{43}$$

Each particle error $e_t^{[m]}$ is calculated given the current range scans in Algorithm 9. Particle pose $\mathbf{s}_t^{[m]}$ has already been propagated for time t, where its occupancy grid $\Theta^{[m]}$ contains range scans through $t - 1$. This model uses several predefined parameters, including the minimum and maximum scan ranges r_{min} and r_{max}, as well as occupancy grid resolution g_{res} and discount factor γ.

Once the measurement model calculates the current map error $e_t^{[m]}$ for each particle, it is finished. The RBPF accumulates decayed error over time, to capture some errors over the length of the path including but not just the most recent measurement.

Algorithm 9 Range Scan Measurement Model

```
for all particles $[m] \in P$ do
  for all scans with range $r_s$ at angle $\theta_s$ do
    if $r_s < r_{min}$ or $r_s > r_{max}$ then
      continue
    end if
    $x_t^q = r_s \cos(\theta_s)$
    $y_t^q = r_s \sin(\theta_s)$
    $\mathbf{q}_t = \mathbf{s}_t^{[m]} + [x_t^q \ y_t^q \ 0]^T$
    $\rho = \lceil r_s / g_{res} \rceil$
    $x^d = (x^q - x_t^{[m]})/\rho$
    $y^d = (y^q - y_t^{[m]})/\rho$
    for $j = 0$ to $\rho + 1$ do
      $x^p = x_t^{[m]} + (j)x^d$
      $y^p = y_t^{[m]} + (j)y^d$
      if $j \leq \rho$ and $\Theta[x^p][y^p]^{[m]} > 0$ then
        increase $e_t^{[m]}$ by $\Theta[x^p][y^p]^{[m]}(\rho + 1 - j)(g_{res}/g_{pr})$
        break
      else if $j > \rho$ and $\Theta[x^p][y^p]^{[m]} < 0$ then
        increase $e_t^{[m]}$ by $-\Theta[x^p][y^p]^{[m]}(g_{res}/g_{pr})$
      end if
    end for
    $e_t^{[m]} = \gamma e_{t-1}^{[m]} + e_t^{[m]}$
  end for
end for
```

3.5.3 Map Construction.

Before FastSLAM computes the weighting and resamples, each particle $[m]$ applies the current set of ranges to its map $\Theta^{[m]}$ [33]. This process is similar to the structure of the measurement model, as it calculates Equations 37, 39, and 40, then steps along each scan. It limits scan ranges to r_{max} when calculating $\mathbf{q}_t$ until it reaches the interior loop. When applying scans, each occupancy grid value is decreased along the path until the scan stops at $\mathbf{q}_t$, where it increases the occupancy value. The process for applying the current range scan to each map $\Theta^{[m]}$ is shown in Algorithm 10.

Algorithm 10 Map Construction

```
for all particles $[m] \in P$ do
  for all scans with range $r_s$ at angle $\theta_s$ do
    if $r_s > r_{max}$ then
      $x^q = r_{max} \cos(\theta_s)$
      $y^q = r_{max} \sin(\theta_s)$
    else if $r_s > r_{min}$ then
      $x^q = r_s \cos(\theta_s)$
      $y^q = r_s \sin(\theta_s)$
    end if
    $\mathbf{q} = \mathbf{s}_t^{[m]} + [x^q \ y^q \ 0]^T$
    $\rho = \lceil r_s / g_{res} \rceil$
    $x^d = (x^q - x_t^{[m]})/\rho$
    $y^d = (y^q - y_t^{[m]})/\rho$
    for $j = 0$ to $\rho + 1$ do
      $x^p = x_t^{[m]} + j \ x^d$
      $y^p = y_t^{[m]} + j \ y^d$
      if $j \leq \rho$ then
        decrease $\Theta[x^p][y^p]^{[m]}$ by $(4/g_{pr})$
      else if $j > \rho$ and $r_s < r_{max}$ then
        increase $\Theta[x^p][y^p]^{[m]}$ by $(8/g_{pr})$
      end if
    end for
  end for
end for
```

Instead of measurement model error calculations, the particle decreases its $\Theta^{[m]}$ cell values along each coordinate $\mathbf{p}$ and increases its cell value at the last scanned coordinate $\mathbf{p}$. Each cell has g_{pr} possible occupancy values, so the grid requires several scans to indicate a clear value. After Algorithm 10, cell values correspond to a likelihood of being occupied.

This algorithm is the most significant contribution to the computation time of the FastSLAM implementation. This is due first to the loop for each time step for each particle for each range scan. This is the same requirement as the measurement model, but applying scans to maps also requires an additional loop as the algorithm walks down the length of each scan. At each point, one cell of the current particle map $\Theta^{[m]}$ must be accessed and written, resulting in additional time for the significant memory requirement. The FastSLAM creators have presented a method to reduce this requirement by sharing map storage possible through resampling, though this is not implemented here [46].

Because there are M particles, the set of all maps represents Θ in the same way the poses of P represent the pose distribution. As some particles are more accurate than others, FastSLAM weights and resamples P to achieve a better distribution over its current pose $\mathbf{s}_t$ and map Θ.

3.5.4 Particle Weighting.

Before the RBPF can resample, it must assign a weight $w_t^{[m]}$ to each particle $[m]$ at time t. The RBPF is compatible with any method used to weight each particle, as long as the weights are normalized such that they sum to 1. FastSLAM begins by subtracting the minimum values from each map error $e_t^{[m]}$ as calculated by the measurement model. In doing this, it distinguishes small errors from large errors appropriately. Their difference results in a total particle error $\epsilon_t^{[m]}$ shown in Equation 44.

$$\epsilon_t^{[m]} = \left(e_t^{[m]} - \min(e_t)\right) \tag{44}$$

To ensure a balance of low error particles, FastSLAM limits $\epsilon_t^{[m]}$ to a minimum value of 2. This prevents particles from having an error much lower than the rest and a resultant undesired spike in the posterior distribution. Limiting the error value, and thus the weighting, discourages the resampling algorithm from replacing too many particles with a copy of the same single particle.

FastSLAM then creates the weight $w_t^{[m]}$ for each particle from $\epsilon_t^{[m]}$. Equation 45 calculates the weight of each particle, taking the reciprocal of $\epsilon_t^{[m]}$ and normalizing the result. This process assigns each particle $[m]$ a weight $w_t^{[m]}$ at time t.

$$w_t^{[m]} = \frac{1}{\epsilon_t^{[m]}} \left(\sum_{m=1}^{M} \frac{1}{\epsilon_t^{[m]}} \right)^{-1} \tag{45}$$

Once FastSLAM weights the set of particles P, the final step is to resample the particle distribution. FastSLAM uses an adaptive resampling technique as done in similar research successful with grid mapping [29]. FastSLAM executes a test after each weighting to determine if a resampling step is needed. This not only reduces the required computation, but also reduces the chances of a good particle being replaced from resampling too often.

FastSLAM tests for resampling using each particle weight $w_t^{[m]}$. It calculates the number of effective particles m_{eff} by determining a sum of the squared weights. The weights are normalized, but their squares allow FastSLAM to find how different the highly weighted particles are from the lower ones. Equation 46 describes the calculation for number of effective particles m_{eff}.

$$m_{eff} = \frac{1}{\sum_{m=1}^{M} \left(w_t^{[m]}\right)^2} \tag{46}$$

FastSLAM then performs the resampling test. If m_{eff} is less than half the particles $M/2$, then it carries out the resampling process. If m_{eff} is greater than $M/2$, FastSLAM does not resample P and all steps are finished for time t.

3.5.5 Resampling.

FastSLAM uses the resampling process to adjust its particle distribution. This implementation uses a standard SIR process in a way that is both nonlinear and non-Gaussian [27]. It begins by finding a normalized cumulative sum of likelihood weights w_{cum} using the Kahan algorithm designed for this purpose [37]. It then chooses a particle once for each time its cumulative weight intersects with a random draw $\mathbf{u}_r$ from a uniform distribution $\mathcal{U}$ from 0 to 1. The algorithm carries a $O(M \log M)$ complexity from sorting u_r.

With the number of times to resample each particle p_{rs}, FastSLAM replaces each state $\mathbf{x}$ and map Θ for lower weighted particles with those from higher weighted ones. The total process for updating the RBPF is shown in Algorithm 11.

Once FastSLAM resamples particle set P it updates the RBPF, completing all necessary FastSLAM steps. An attractive aspect of an online SLAM implementation is that a mapped solution makes itself available at any time t. Here, FastSLAM provides $\mathbf{x}^{[m]}$ and map $\Theta^{[m]}$ at all times, where particle $[m] \in P$ has the lowest error $\epsilon_t^{[m]}$ and thus highest weight $w_t^{[m]}$.

3.6 Summary

With all calculations at time t complete, MINS and FastSLAM proceed to the next time $t+1$ accepting the subsequent sensor inputs. The next time step begins by reading IMU data, integrating its accelerations to velocities, removing the angular bias, then integrating velocities to IMU pose change $\delta\mathbf{s}_{t+1}^{imu}$. This pose change pro-

Algorithm 11 RBPF Resampling

```
$w_{cum},\ c,\ j,\ u_s,\ u_{rs} = 0$
**for all** particles $^{[m]}$ **do**
    $y = w^{[m]} - c$  // *Kahan algorithm*
    $t = w_{cum} + y$
    $c = t - w_{cum} - y$
    $w_{cum} = w^{[m]} = t$
**end for**
$\mathbf{u}_r = M$ sorted samples from $\mathcal{U}(0,1)$
$\mathbf{u}_r = (w_{cum})\mathbf{u}_r$
**for all** particles $[m]$ **do**
    $p_{res} = 0$  // *assume not resampled until found*
    **if** $u_s < M$ **and** $u_r^{[u]} < w^{[m]}$ **then**
        increment $u_{rs}$
        **repeat**
            increment $p_{res}$ and $u_s$  // *count resamples*
        **until** $u_s < M$ **and** $u_r^{[u]} < w^{[m]}$
    **end if**
    $p_{rs}^{[j]} = p_{res}$
    increment $j$
**end for**
$j = M$  // *update particle set based on resampling*
**for** $i = M$ **to** 1 **do**
    **if** $p_{rs}^{[i]} > 0$ **then**
        decrement $j$
        $\mathbf{x}^{[j]} = \mathbf{x}^{[i]}$
        $\Theta^{[j]} = \Theta^{[i]}$
    **end if**
**end for**
**for all** particles $[m]$ **do**
    $k = p_{rs}^{[m]}$  // *replicate live samples*
    **if** $k > 0$ **then**
        **repeat**
            $\mathbf{x}^{[i]} = \mathbf{x}^{[j]}$
            $\Theta^{[i]} = \Theta^{[j]}$
            increment $i$, decrement $k$
        **until** $k > 0$
        increment $j$
    **end if**
**end for**
```

vides the prediction update the linear Kalman filter maintaining position state $\mathbf{x}_{t+1}$ and covariance $\mathbf{\Sigma}_{t+1}$. This continues until an egomotion or an odometry update is available.

When an image is available, MINS extracts image features from stereo cameras. It then matches features in both images at $t+1$ with those in both images at t, and calculates an egomotion pose difference $\delta\mathbf{s}_{t+1}^{ego}$. The linear Kalman filter accepts $\delta\mathbf{s}_{t+1}^{ego}$ as an observation update. When an odometry update is available, MINS measures the odometry difference as $\delta\mathbf{s}_{t+1}^{odo}$ directly from the sensor. It provides $\delta\mathbf{s}_{t+1}^{odo}$ to the linear Kalman filter as an observation update as it does for an egomotion update.

Once the Kalman filter has been updated, its state $\mathbf{x}_{t+1}$ becomes the current pose of the MINS system $\mathbf{s}_{t+1}$. This research then uses this pose as the control input u_{t+1} and the current LIDAR range scan as the measurement input z_{t+1} to a FastSLAM implementation.

The FastSLAM implementation carries out the odometry motion model using the Kalman filter covariance $\mathbf{\Sigma}_{t+1}$ within its error parameters. This creates a distribution of particle set P, representing M proposed current poses $\mathbf{s}_{t+1}$. FastSLAM then executes a measurement model, incorporating LIDAR measurement scan z_{t+1} and each map $\Theta^{[m]}$ for each particle $[m]$. The measurement model calculates a map error $e_{t+1}^{[m]}$ for each particle, which the FastSLAM converts into a normalized weight $w_{t+1}^{[m]}$. Finally, if m_{eff} does not reach $M/2$, FastSLAM resamples particle set P using the SIR method. This completes the steps for time $t+1$.

Each subsequent step follows the same procedure with new sensor data, mapping the data at each new time t. MINS and FastSLAM continue this cycle until all inputs are exhausted. At this point, FastSLAM produces map $\Theta^{[m]}$ from the particle $[m]$ with the highest weight as its current belief of the environment.

IV. Results & Analysis

This research seeks to execute the implementation on real data gathered by a mobile ground vehicle. The MINS system operates by design on two dimensional data gathered by stereo cameras, an inertial unit, and wheeled odometry. In the same way, FastSLAM desires two dimensional LIDAR scans corresponding to points on the MINS output path.

To accomplish this, a vehicle outfitted with all sensors explores an indoor hallway environment and stores all input data for the implementation. The implementation configures the various algorithms within MINS and FastSLAM to produce paths and occupancy grid maps as the SLAM solution of this research.

This chapter discusses the specifics of the data gathering and values used in experiments. Next, it presents the results of the feature extraction and sensor input paths used as MINS input. Following this, it compares the path output from the MINS system to the similar FV-SIFT navigation system. Lastly, it displays grid maps produced by FastSLAM and discusses their differences.

4.1 Testing Procedure

The test data is collected from the Pioneer P2-AT8 vehicle shown in Figure 12. The P2-AT8 provides internal odometry on skid steering wheels fitted with indoor tires. Mounted on the vehicle are a SICK LMS 200 LIDAR unit, which measures ranges in a 180° horizontal sweep at 1° increments, giving it centimeter resolution at a distance up to r_{max}.

The vehicle also carries the necessary hardware for FV-SIFT to operate. This hardware consists of two PixeLINK PL-A741 machine vision cameras with 1280×960 resolution fitted with Pentax lenses giving a 90° field of view. This is combined

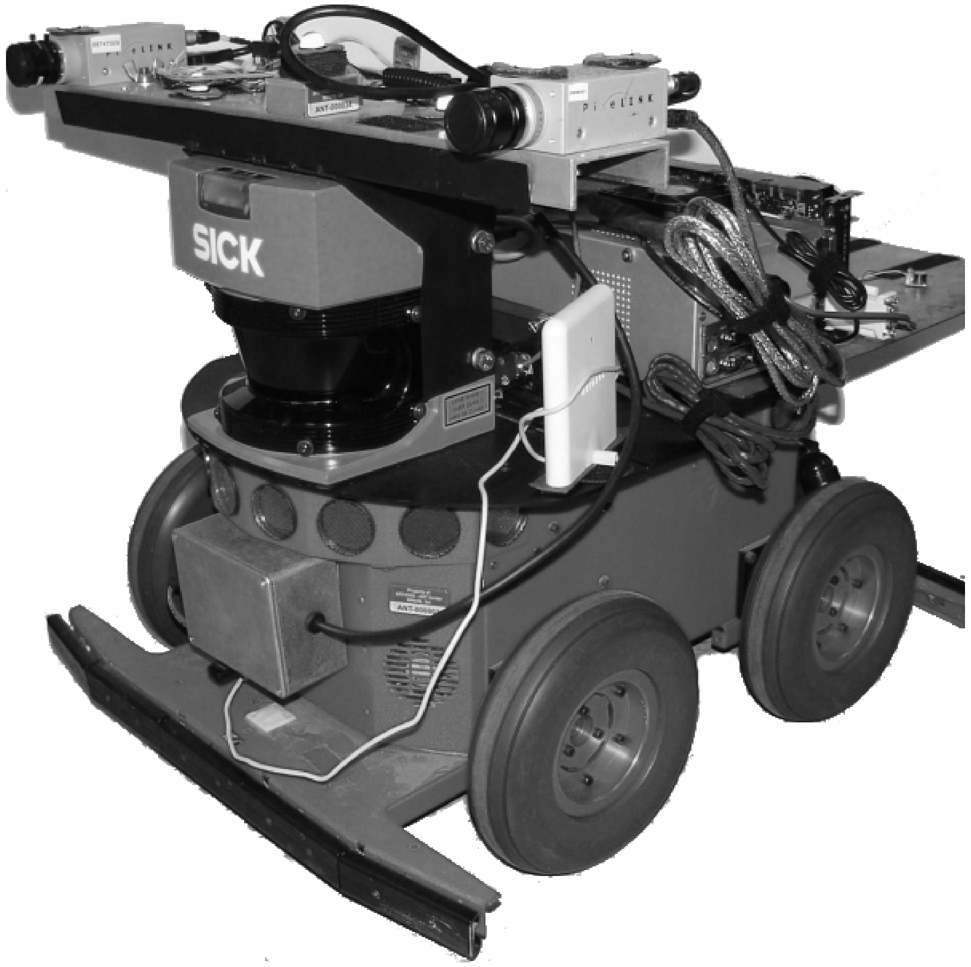

Figure 12. The Pioneer P2-AT8 vehicle used in testing. It carries stereo cameras and an IMU above the LIDAR, with an external computer toward the rear of the chassis.

with a MicroRobotics MIDG II consumer grade MicroElectricalMechanical Systems (MEMS) IMU to provide the measurement data.

Due to the intensive requirements of image processing, an external computer with a 2.0 GHz Core2Duo processor and 4 GB of memory can process the left and right images. The machine is capable of executing a parallelized SURF feature extraction algorithm for FV-SIFT as it collects data by utilizing Single Instruction Multiple Data (SIMD) calculations on an Nvidia 9800GTX+ Graphics Processing Unit (GPU) [64]. This computer configuration records images and IMU data; it is implemented for the FV-SIFT path but not the MINS path presented in the remainder of this chapter.

The internal vehicle machine, a 1.6 GHz Pentium M with 1.5 GB of memory, records odometry and laser ranges simultaneously on a VersaLogic EBX-12 board. The external computer connects to the internal one via Ethernet cable to communicate FV-SIFT output for comparison and to synchronize time between the sensors. This arrangement allows each computer to collect the desired input data into log files for future use.

The vehicle path resulting from a manually driven test run serves as the data set presented in the remainder of this chapter. The vehicle records data for 9 minutes and 47 seconds while both stationary and moving in this run, and for 10 minutes and 42 seconds in the additional data collection run presented at the end of this chapter.

For both runs, the environment consists of an indoor hallway with a tile floor that measures 2.5 meters wide. The vehicle begins in the northeast corner before traveling left around a rectangle approximately 30×40 meters in size. After returning to its starting location, the vehicle makes a right turn into a room and stops, ending the first run.

This run creates a classic SLAM situation of closing a large loop. The sensors must be accurate enough and the number of particles must be high enough to ensure the position may be resolved correctly once the vehicle returns. A highly trusted pose estimate allows the RBPF to use a minimal number of particles to cover a smaller distribution, but a larger loops require more particles to cover a constantly growing distribution. This is true for any FastSLAM implementation, regardless of input sensors available or map storage strategy.

4.2 Implementation Details

The hardware saves all camera images for later use as portable graymap files in Netpbm format, and saves all other collected sensor data in text files. With data from all sensors available, the physical implementation consists of two main phases.

The first phase deals strictly with the image processing, feature extraction, and egomotion within MINS. This process is done using MATLAB® script files and running a compiled executable to extract SIFT features, not yet implemented on the GPU. This produces a text file consisting of egomotion positions.

The second phase reads all text files, implements the linear Kalman filter and FastSLAM, storing the occupancy grid maps as Netpbm graymap format files. This phase is a standalone Win32 application built on the Bayes++ filtering library using Microsoft Visual Studio 2005 in the C++ language.

The implementation reads one set of stereo images, inertial measurements, odometry data, and laser ranges simultaneously. Even though the computers synchronize their clocks, the sensors collect and store data at different rates. Operating at a constant 50 Hz, the IMU is the fastest sensor and updates the Kalman filter prediction at this frequency.

The vehicle stores its odometry and LIDAR log files at an average of 12 Hz. MINS first provides its Kalman filter with an odometry observation at this frequency. Because the measurement model applies the range scan, FastSLAM then performs the sequential steps described in the previous chapter. To restate, MINS predicts its Kalman filter with each IMU measurement until there is an odometry reading available, when it observes the Kalman filter and continues with one time step of FastSLAM operation.

Because of the time required to calculate and save them, images are only available at 2 Hz. While the IMU and odometry predict and observe for the linear Kalman filter at a higher frequency, the feature extraction and egomotion algorithms only update the Kalman filter when available. The filter design prevents this difference in update frequencies from adversely affecting its operation.

4.3 Algorithm Parameter Settings

The methodology discussed many values that either model the hardware used or affect the output as desired. Thus, the MINS and FastSLAM implementations contain many unique parameters, which must be set. These include models of the cameras, egomotion settings, inertial integration settings, and parameters for the motion and measurement models within FastSLAM.

4.3.1 Camera Model.

The model of the stereo cameras contains many values specific to the hardware used in the feature tracking and egomotion implementation. All are properties of the model and settings of the cameras on the left L and right R sides. This research does not seek to modify the camera model, as it does not change any hardware settings [23, 65].

These values include the $n_C \times m_C$ camera pixel resolution and offset $\mathbf{o}_C$ from the central IMU position. It also includes the principal point $\mathbf{p}_C$ and focal length $\mathbf{f_C}$ vectors, each with an x and y value, and the five CalTech distortion model values $\mathbf{k}_C$ without units. Square matrices $\mathbf{C}_C$ and $\mathbf{T}_C$ are both Direction Cosine Matricies (DCMs) in three dimensions used to relate image position to a coordinates as done in the previous chapter. Table 1 provides the camera model parameters used in the feature tracking algorithms.

All values in Table 1 are unchanged from earlier findings [65]. The models for the left L and right R cameras include the same parameters, and each is used to find the feature locations from each stereo image pair.

Table 1. Camera Parameters

Parameter	Value	Units
m_C	960	pixels
n_C	1280	pixels
$\mathbf{o}_C$	[0.0 -0.442 0.0]	meters
$\mathbf{p}_C^L$	[647.0628 483.6622]	pixels
$\mathbf{f}_C^L$	[698.0197 697.9811]	pixels
$\mathbf{k}_C^L$	[-0.283 0.0778 -0.000563 -0.00132 0.0]	none
$\mathbf{C}_C^L$	$\begin{bmatrix} 1.0 & -0.0025 & 0.0 \\ 0.0025 & 1.0 & -0.006 \\ 0.0 & 0.006 & 1.0 \end{bmatrix}$	meters
$\mathbf{T}_C^L$	$\begin{bmatrix} -593.224 & 0.0 & 485.2222 \\ 0.0 & 531.882 & 643.0426 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$	pixels
$\mathbf{p}_C^R$	[618.1513 476.1207]	pixels
$\mathbf{f}_C^R$	[698.8774 699.5389]	pixels
$\mathbf{k}_C^R$	[-0.2888 0.0817 -0.00166 0.000128 0.0]	none
$\mathbf{C}_C^R$	$\begin{bmatrix} 0.99988 & 0.00539 & -0.01474 \\ -0.00573 & 0.9997 & -0.02314 \\ 0.0146 & 0.02323 & 0.9996 \end{bmatrix}$	meters
$\mathbf{T}_C^R$	$\begin{bmatrix} -592.8576 & 0.0 & 472.558 \\ 0.0 & 535.8046 & 624.392 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$	pixels

4.3.2 Egomotion Model.

In addition to existing hardware models, the previous chapter also contains several manually set parameters. Each affects a different aspect of MINS operation in the egomotion calculations. This includes the ratio ϱ between the best and second best feature matches, the maximum allowable angle difference ϑ for a feature match, and the fraction of standard deviations σ_l a feature location may move within to remain a tracked landmark. Table 2 lists each of these values.

Table 2. Egomotion Parameters

Parameter	Value	Units
ϱ	0.6	none
ϑ	1.5	deg
σ_l	0.75	m

Each of these values was manually tuned to remove all detectable false matches in sample images, and maintain at least one tracked feature between each image pair. These parameters result in the egomotion path used as an observation input to the MINS linear Kalman filter.

4.3.3 Inertial Model.

For inertial integration, the IMU operates at a constant frequency of 50 Hz. Thus, MINS models this with time period dt as the it processes each IMU measurement. The vehicle maximum velocity v_{max} is known by the motor settings and the vehicle operator while collecting data, so MINS uses parameter in the inertial integration. The IMU standard deviations of its internal accelerometer σ_{acc} and gyroscope σ_{gyro} for use within the Kalman filter are taken from the physical specifications of the unit [65]. Table 3 lists each of these values as used in the previous chapter.

Table 3. Inertial Parameters

Parameter	Value	Units
dt	0.02	s
v_{max}	1.0	m/s
σ_{acc}	0.196	m/s^2
σ_{gyro}	0.0087	rad/s

These parameters result in an integrated IMU path that provides the linear Kalman filter prediction. This covers all parameters used through MINS operation.

In comparison to the egomotion and inertial parameters, the odometry hardware measures its path directly and thus does not require any processing. MINS does not model odometry error outside of the settings discussed in Chapter 3. The error in combined input path u_t is instead handled in FastSLAM by the motion model.

4.3.4 Motion Model.

Any FastSLAM solution contains several parameters that change calculations in the motion and measurement models. In the motion model, the only parameters are the four values, a_2 and a_4 directly, with a_1 and a_3 modified by the covariance values. Table 4 displays the four motion model parameters, with units converting distances to headings and back.

Table 4. Motion Parameters

Parameter	Value	Units
a_1	0.005	rad/rad
a_2	0.001	rad/m
a_3	1.0	m/m
a_4	0.05	m/rad

The operator changes these motion values to model the believed error in the vehicle odometry. In this research, control input u_t is the MINS system path, so these parameters instead model the error of this system. The researcher selected these values to attain the desired particle distribution given the error of the MINS path.

4.3.5 Measurement Model.

The measurement model works with any range scan and any size grid. The LIDAR range minimum r_{min} and maximum r_{max} come directly from the hardware specifications. However, the researcher sets occupancy grid values depending on the scale of the data. In these results, the grid is $g_x \times g_y$ meters in size with resolution g_{res}. Within each cell, the occupancy grid stores a value within the range g_{pr} as a probability of being occupied. This and discount factor γ do not have units. Table 5 provides each of the parameters used in the FastSLAM implementation measurement model and map builder.

Table 5. Measurement Parameters

Parameter	Value	Units
r_{min}	0.01	m
r_{max}	8.10	m
g_x	70.0	m
g_y	70.0	m
g_{res}	0.1	m
g_{pr}	256	none
γ	0.99	none

Because r_{min} and r_{max} are specific to the sensor, the grid is the only entity that affects the measurement model parameters. As discussed later in this chapter, the occupancy grid settings depend primarily on computer memory available. This grid resolution g_{res} value was previously identified as the best tradeoff between detail and effectiveness [61]. Discount factor γ values are typically large fractions, with this value experimentally determined to result in the desired total error values [52]. Using these parameters, FastSLAM produces the resulting maps presented in this chapter.

4.4 Path Comparison

All three linear Kalman filter input paths; stereo image egomotion, inertial integration, and vehicle odometry, are displayed in Figure 13. This visualizes the strengths and weaknesses of each sensor and supports the efforts of this research to combine the three paths.

The path obtained by egomotion does not initially appear to be particularly accurate. Nonetheless, it displays motion when expected and travels in the known direction when it tracks a reasonable number of features. When the vehicle reached the southeast corner after traveling three quarters of the loop, it tracks one to four features until facing north again. This results in a large discrepancy and is a visible part of the egomotion path.

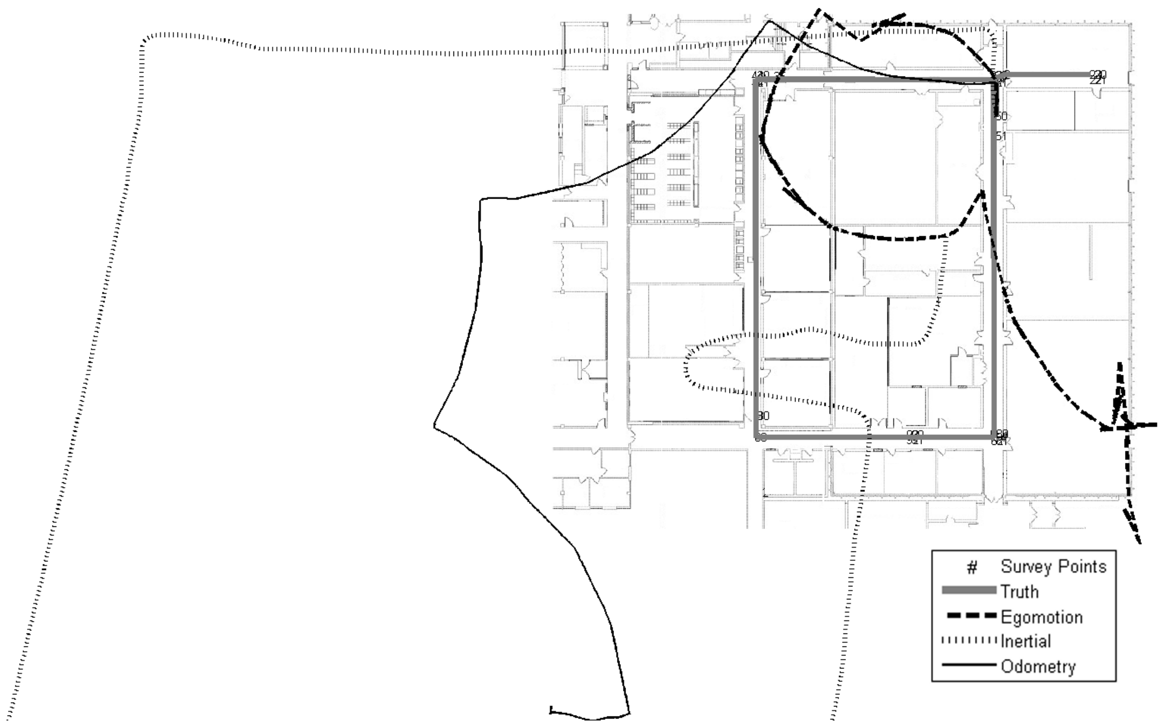

Figure 13. Egomotion, inertial integration, and odometry paths. They are plotted over the building floor plan with an approximate true path for reference.

The IMU measures direction and overall motion with surprising consistently. However, it is much less capable of measuring distances as the integration results in a slight exaggeration of velocity and a larger one in distance. Thus, the inertial path is much larger than that actually traveled but appears similar in shape. This path also contains a significant error, once the vehicle returns to the northeast corner. The integration translates stationary rotations to the left then back to the right as motion, resulting in false motion at this point in its path.

The odometry path appears to be the opposite to the inertial path. Whereas by its nature it measures a very accurate distance traveled, it is subject to compounding errors in heading. An uncorrected balance problem with the drive motors and steering control of this specific test vehicle results in a pull to the left while driving forwards. As a result, the odometry measures motion in a straight line while the vehicle keeps a slight turn. The manufacturer provides a means to compensate the wheel encoders and configure the odometry for this, but this configuration was inoperative and unavailable. The operator was forced to repeatedly correct the vehicle to the right as it drove through the hallways, resulting in the path shown.

Knowledge of the sensor characteristics allows MINS to configure its linear Kalman filter values to take advantage of the desired aspects of each sensor. The properties discussed are as a result of the general sensor environment and the implementation used. Because of this, they are not likely to change from one test to another and show promise in future tests.

MINS provides the three motion estimates from previous time $t-1$ to current time t as input to its linear Kalman filter. After it processes each observation as presented in the previous chapter, its state $\mathbf{x}_t$ produces the next step in the MINS path and the FastSLAM control input u_t. Figure 14 displays the MINS path alongside that from FV-SIFT [23, 65], which operates in real-time using an EKF, but cannot access odometry as MINS does.

Both MINS and FV-SIFT output values are suitable for navigation requirements in local environments without using GPS. However, the differences between the paths are worth noting. For this particular data set, environment noise affects the images early in the path near the northeast corner. FV-SIFT tracks a limited number of twelve image features, and with these covered, can only rely on the IMU for input until it can rediscover features. It recovers from this error, but not after recording a jagged, inaccurate path undesirable for a SLAM solution.

The MINS system presented in this research does not suffer the same consequences given in the same input, as consistent odometry provides an additional input that takes over the linear Kalman filter with the smallest variance. This result is also displayed in other corners, where IMU integration measures false motion and feature tracking finds a new set of features.

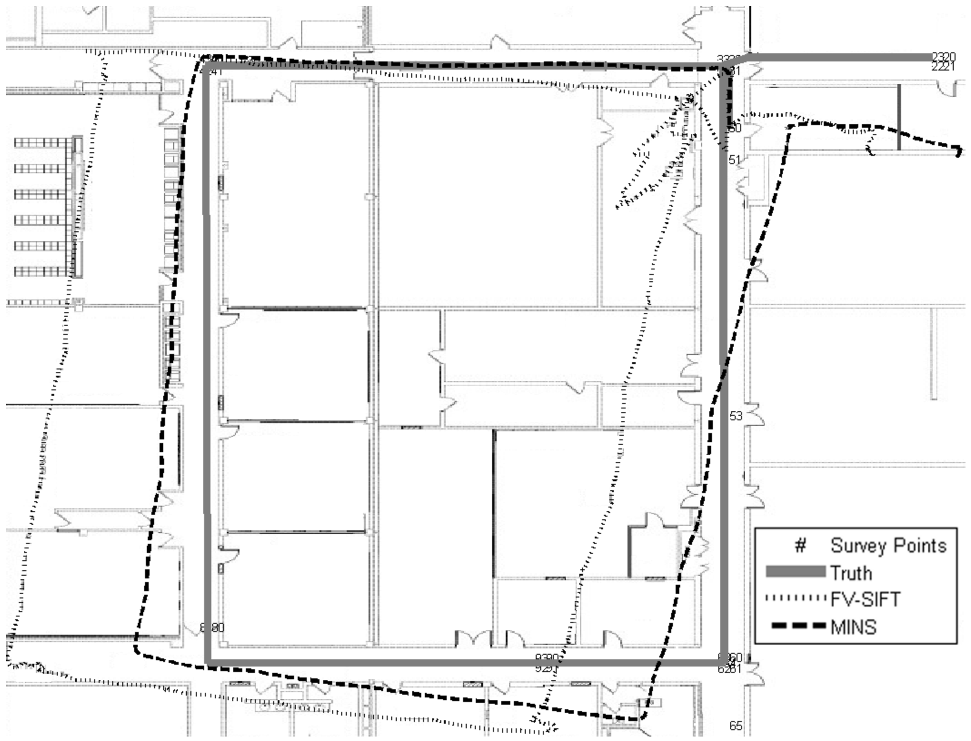

Figure 14. Paths from the MINS system and the FV-SIFT program. The background is a building floor plan with approximate truth shown for reference.

Comparing the two paths to truth is the only way to obtain an error, as the survey points are the only trusted data. The path start, finish, and each of the corners correspond to numbered survey point locations within the hallway. The distance between the current pose and the survey point gives the path error at that location and time. Table 6 displays this error, in meters, at each known point, and includes a mean of the distances.

Overall, the MINS path finished almost a half meter farther away than the FV-SIFT one, but maintains a pose closer to truth through most of its path. The errors in Table 6 do not include errors while traveling along hallways, including the largest discrepancy in the existing system after the first pass of point 34.

When taking the northwest corner, the FV-SIFT path is over 4 meters away, and approximately 10 meters from truth for the duration of the southern hallway. In fact, its low error upon returning at the last two survey points is merely a coincidence of

Table 6. Navigation Filter Error

Map Location	Survey Point	FV-SIFT error (m)	MINS error (m)
Start	50	0.0	0.0
NE	34	0.548	0.261
NW	44	4.429	0.482
SW	83	10.003	3.716
SE	64	9.788	5.413
NE	34	2.931	5.311
Finish	23	5.709	6.185
mean		4.772	3.052

the path taken, and it could easily be farther away having taken another route.

As a comparison, the MINS path is well within one meter of truth through point 44. Its error appears to grow slowly over the course of the path, and finishes just over 6 meters from the final point. A mean of the point errors displays the novel path improves on the existing system by over 1.5 meters, or 35 percent.

4.5 Map Comparison

Mapping uses the linear Kalman filter state and covariance as the control input for use alongside the LIDAR range measurements in FastSLAM. Therefore, MINS exists to provide FastSLAM with as accurate a pose estimate as possible, to reduce the computation required to maintain many particles within the RBPF.

The purpose of the RBPF is to store the number of particles and associated maps required to account for error in the control input. In previous research, the number of particles is of great significance [47]. How it represents each of the particles and maps is also important, as long paths desire a smaller number of particles to limit computation time [28].

4.5.1 The Occupancy Grid.

The $g_x \times g_y$ meter grid with resolution g_{res} is just large enough to capture the hallway size. An important concern of this implementation is how to represent the grid effectively while limiting the memory required. Equation 47 describes the total memory required for M particle maps, where b is the number of bytes required for each grid cell.

$$(M)(b)\frac{g_x \times g_y}{(g_{res})^2} \tag{47}$$

This implementation first limits the map size by representing each cell as the smallest basic data type available. This is a single byte (8 bit) character, leaving $2^8 = g_{pr}$ values available for occupancy. Substituting $b = 1$ and the grid values from Table 5 into Equation 47, each occupancy grid requires 70×70 over 0.1^2 bytes of memory. Table 7 displays the memory requirements for the number of particles M given by powers of ten.

Table 7. Map Memory Requirements

M	memory
1	479 KB
10	4.67 MB
100	46.7 MB
1000	467 MB
10000	4.56 GB

Many particles still causes the memory requirement to grow out of feasibility for such a small map. Even for 1000 particles, the total memory requirements for the implementation remain on the order of megabytes. This amount of memory is hardly a problem for modern computers. However, this means increasing the number of particles to 10000 results in gigabytes of memory. Modern computers can handle this much storage, but FastSLAM requires all maps to be readily accessible. This either requires an expensive machine with significant memory in hardware or significant increase in computation time as memory swaps.

Each cell of the occupancy grid contains a value that corresponds to the probability that cell is occupied. As the majority of cells are never observed, the majority of maps is typically an medium shade of gray. As the vehicle scans its environment, FastSLAM decreases the probability of the grid cells it scanned through and increases those where the scan stopped. The result is white space between black walls, with the area behind it medium gray until the vehicle explores the area.

4.5.2 Output Comparison.

FastSLAM builds a map from any path and range scan it receives as input. The occupancy grid map is an easily seen indication of accuracy, but it is difficult to compare quantitatively without comparing the path as done in the previous section [10]. This is the case even after considering a visual comparison of the grid to a floor plan, if a trusted one exists.

An occupancy grid image is straightforward to measure and easily interpreted by an operator. It clearly displays hallways, rooms, obstacles, doorways, and even closing doors [61]. The maps generated using this implementation clarify the problems with a path, as heading errors are visible through applying range scans.

To illustrate the relative accuracy of the input paths, the implementation produces maps given different possible input paths and the same range scans. Figures 15, 16, and 17 display maps obtained given the various paths discussed from odometry, FV-SIFT, and MINS, respectively. This implementation generated these maps without using the RBPF to generate multiple noisy particles. This section compares these maps to the FastSLAM output in Figure 20.

The odometry map in Figure 15 reflects the steering error of the wheels. Accurate

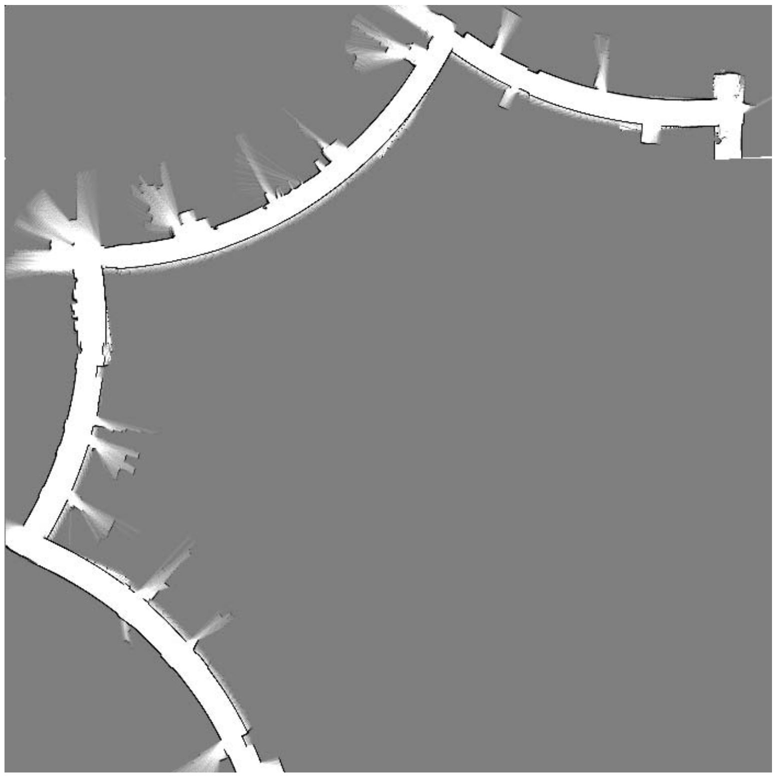

Figure 15. Map produced with the odometry path and range scans. The structure of the hallway is clearly visible, but it appears skewed.

with respect to distance, its heading curves the hallways off the area. This would be the input for a typical SLAM solution using only odometry.

It is theoretically possible for a RBPF to create an exact path from this odometry input. However, it would require precisely tuned motion model α values to account for the enormous odometry error. With accurate settings, such large parameter values would most likely demand millions of particles to represent a large enough distribution. Any implementation using this number of particles would require an unreasonable amount of calculation time to produce a consistent path.

FV-SIFT displays a more accurate heading in Figure 16. It suffers from different problems not affecting the odometry path, such as each of its corners being uncertain. Most significantly, the error in the northeast corner produces a false open area which actually corresponds to the northern hallway. This error propagates to the rest of the

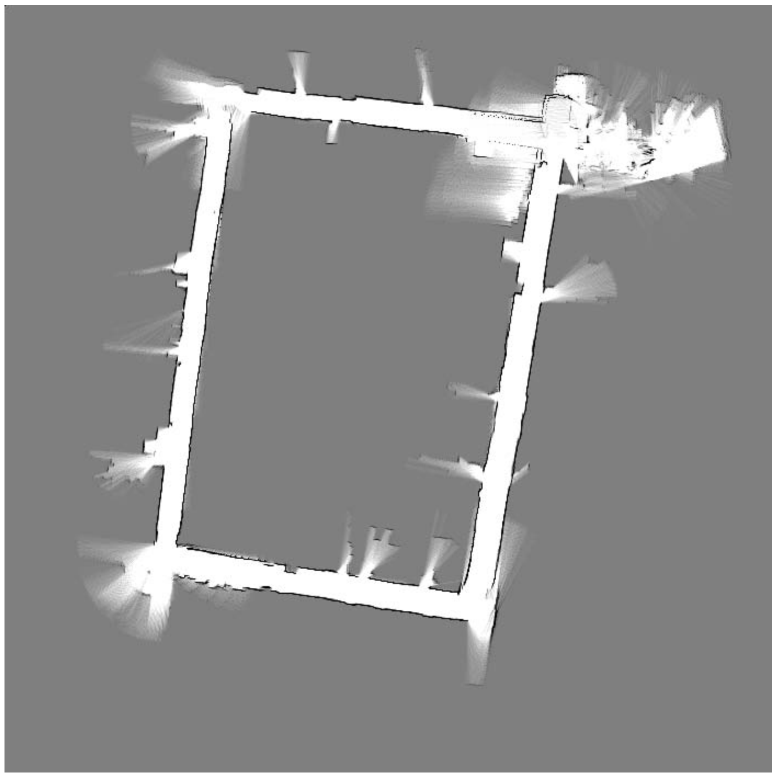
Figure 16. Map generated with the FV-SIFT path. It displays straight, slanted hallways due to significant inaccuracies in the corners, especially the northeast [23, 65].

path, even if it returns close to the starting location.

The MINS path in Figure 17 is more believable. This reflects greater confidence in the location of walls and corners, and is the desired result from a filtered solution like FastSLAM. Slowly accumulating inaccuracies move the angles of later hallways from closing the loop, so this path is also not free of errors.

These maps show the MINS path is far superior to a solution that uses just the odometry. It is not subject all the problems of the existing system, and maintains a much more accurate heading than skid steering odometry. The advantage of accurate sensors such as the LIDAR and a good path is the ability to reduce the number of particles and be able to process paths on loops such as these hallways.

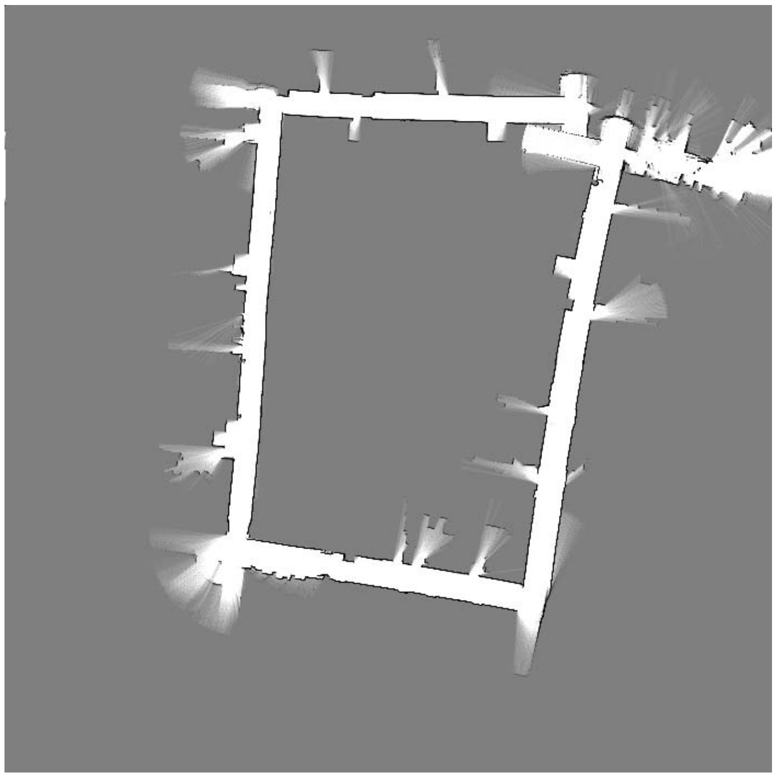

Figure 17. Map generated using the MINS path. It displays more accurate hallways than odometry, more accurate corners than FV-SIFT, and is close to closing the loop.

4.5.3 Particle Filter Output.

The design of the RBPF within a FastSLAM implementation introduces controlled noise to correct its input path. When given control input u, each particle $[m] \in P$ carries a slightly different path from that of the input from the motion model. The RBPF uses its measurement model to adjust the error and weights of particle set P before it resamples. This maintains particles whose path matches closest to range scan observations z.

An IBM laptop computer with 2 GHz Intel® Pentium® M processor and 2 GB of memory executes the second phase of the implementation, including the MINS and FastSLAM algorithms. As FastSLAM scales linearly with respect to the number of particles, this machine takes 4 to 5 minutes for every 10 particles used in executing the 10 minute data set.

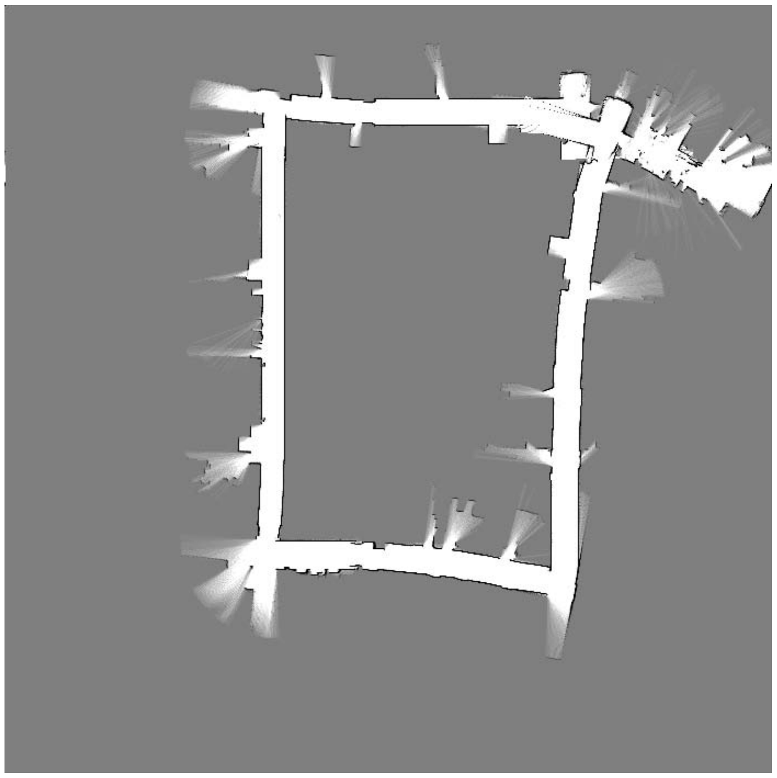

Figure 18. Map from the FastSLAM implementation using odometry as input. It straightens the crooked hallways but not enough to be accurate or close the loop.

The final FastSLAM output is the map from the particle carrying the highest weight, thus the particle most accurate to the true path. For comparison, when control input u comes from odometry, FastSLAM produces the map in Figure 18. FastSLAM generated this map using 100 particles.

The effects of the FastSLAM motion and measurement models improves the odometry path significantly. However, generating this map requires a larger number of particles to represent the odometry error, and FastSLAM cannot remove the rightwards curve of the hallways. Thus, the path returns close to the starting location but is unable to accurately close the loop.

The FV-SIFT path is more accurate in position than the odometry path, but is also less consistent. To illustrate these effects, the FV-SIFT path is provides its path to the FastSLAM implementation. When FastSLAM takes its control input u from

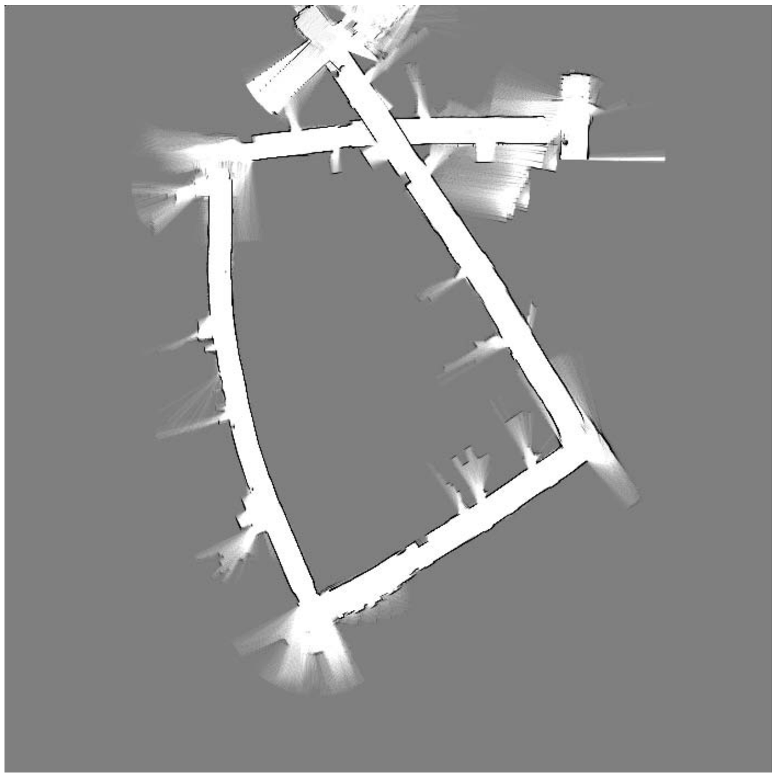

Figure 19. Map from the FastSLAM implementation using the FV-SIFT path as input. Errors in the input path carry though FastSLAM with disastrous effects.

the FV-SIFT output path, it produces the map in Figure 19, also using 100 particles.

This map suffers from the FV-SIFT path errors in the northeast corner already discussed. However, this error is much more difficult to predict than odometry error as done in the motion model. The result is that FastSLAM is unable to correct these errors and they result in a map that is less accurate than the input path depicted in Figure 16.

Using the additional sensors in the MINS system improves the control input u provided to FastSLAM as presented in this research. Figure 20 displays the map generated by this implementation using all parameters and settings presented. FastSLAM generated this map using only 30 particles.

This map is not perfect, most evident by the difference in the northeast corner as the vehicle completes the loop. Unfortunately, early errors in the path adversely affect

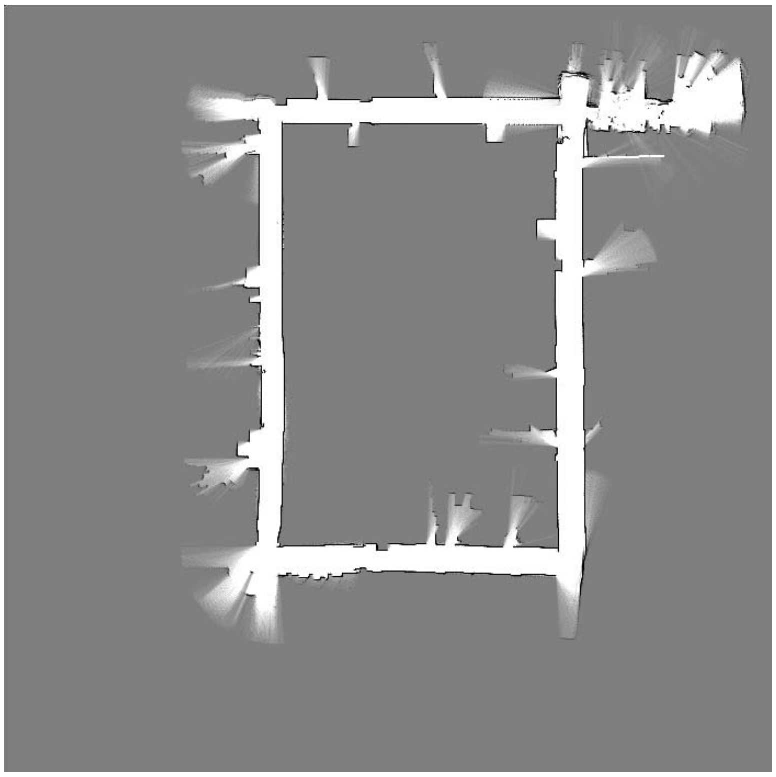

Figure 20. Map from the highest weighted particle from the FastSLAM implementation using MINS as input. It places its hallways straight and returns to the initial location.

each pose afterward, however subtle they are. However, the FastSLAM algorithm using the MINS path closes the loop within a meter of error.

Even without being perfect, the FastSLAM output is a noticeable improvement to the MINS output used to produce the map in Figure 17. Most importantly, FastSLAM successfully matches scanned walls as it returns to the northeast corner. Both the MINS position and the odometry FastSLAM position at this same time is farther away and at an angle, failing to close the loop. Table 8 compares the current best particle path from the odomery FastSLAM and the MINS FastSLAM implementations. Both path errors are measured from the same hallway survey points displayed earlier.

At each point, the output path from the FastSLAM implementation decreases the error of its input. The position of the best current particle is accurate to within a meter at all measurable locations, with an average under half of a meter from

Table 8. FastSLAM Error

Map Location	Odometry FastSLAM error (m)	MINS FastSLAM error (m)
Start	0.0	0.0
NE	0.359	0.269
NW	0.128	0.584
SW	0.898	0.742
SE	2.452	0.863
NE	3.709	0.096
Finish	6.000	0.298
mean	1.935	0.408

the survey points. Using the MINS path not only requires 30% of the odometry FastSLAM computation time by reducing the number of particles, but improves on its path by reducing the error by 79 percent.

4.5.4 Additional Testing.

To validate both the MINS system and the FastSLAM implementation, the testing procedure provides another data set collected by the same vehicle. All algorithm settings are held to the same values, making the only change the input data. This testing ensures the parameter settings and models used are not specific to one data set and can be extended to other situations.

In this additional data, the same vehicle traveled the same environment taking a different path. The vehicle begins in the same starting location and completes the same loop. Instead of then turning off into the room, the vehicle begins another loop retracing its path along the hallways before finishing in the southeast corner. This collection lasts 10 minutes and 42 seconds due to farther distance traveled and less time stopped.

The paths measured by egomotion, IMU integration, and odometry are similar to the other data. The MINS path is also comparable as it uses these as input, but

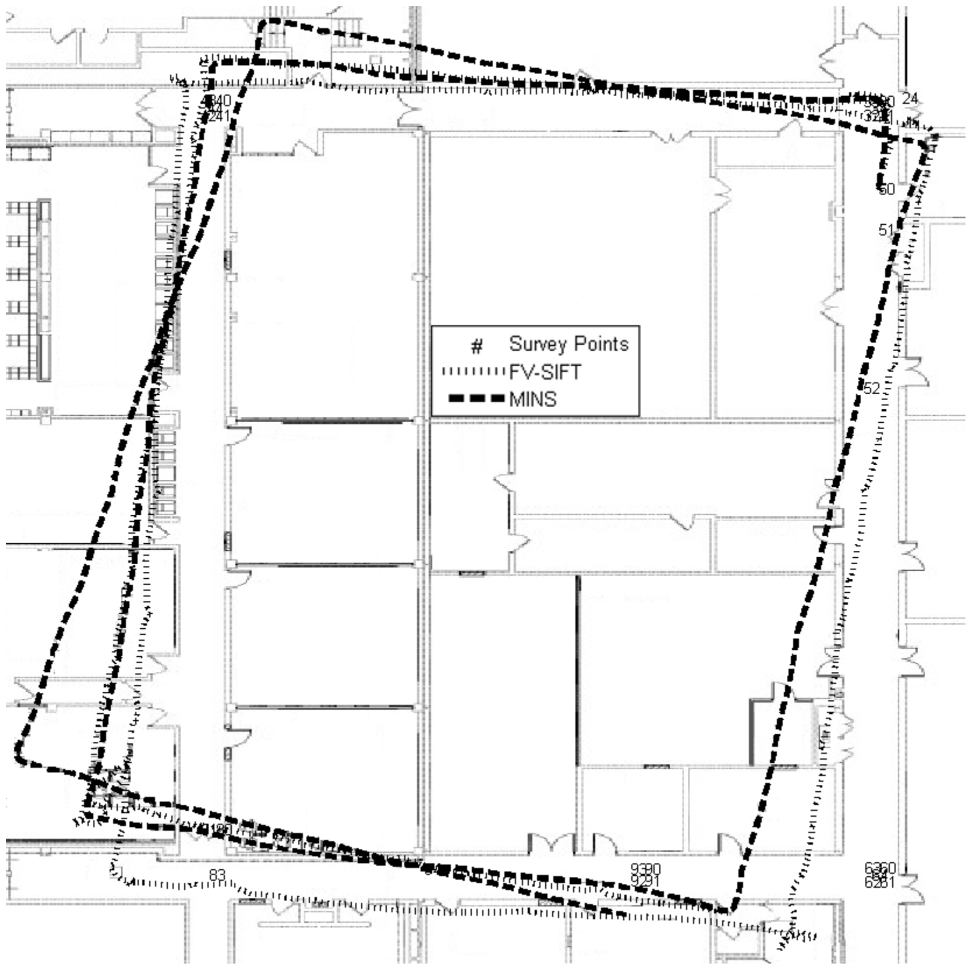

Figure 21. Paths from the MINS system and the FV-SIFT program on the additional data set, shown over the building floor plan. FV-SIFT provides a more accurate path, but MINS provides a smoother path.

FV-SIFT is notably more accurate as it avoids the severe feature loss it experienced in the previous run. Figure 21 shows the paths generated from the FV-SIFT and MINS systems over the same floor plan.

There is no loss of features in the first corner, so the FV-SIFT path is closer to truth than the MINS path at most points. This is not indicate a failure of the MINS system, however, as closer inspection reveals that the FV-SIFT path is still more jagged than the MINS path. This is often accepted for a navigation solution, but results in significant motion model noise and thus a difficult path when used as SLAM input.

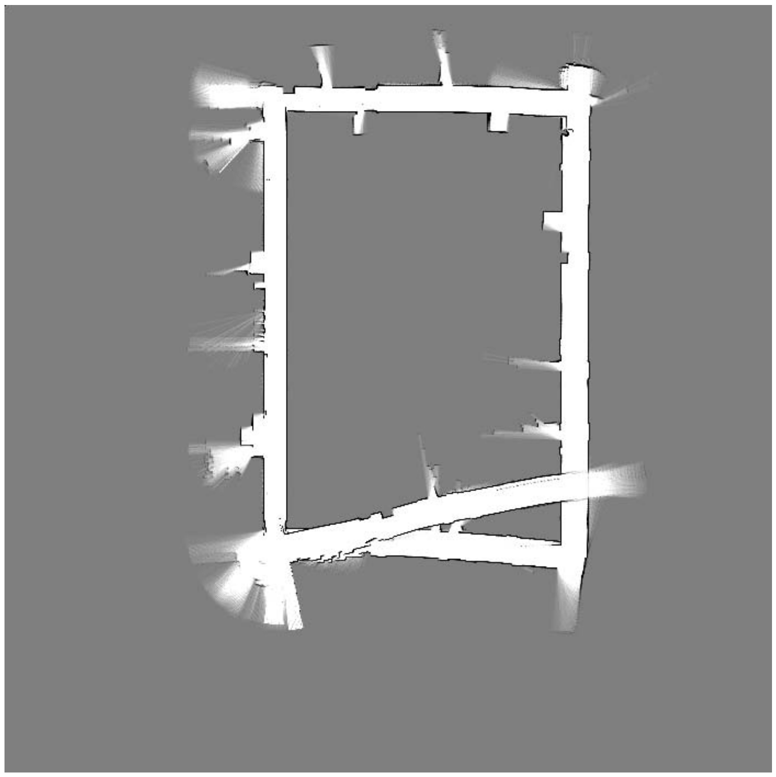

Figure 22. Map from the highest weighted FastSLAM particle using the additional data MINS path as input. It completes the loop then continues matching the map to previously traveled environment.

Using the MINS path for FastSLAM input produces another map of the same area from the additional data set. Figure 22 displays the map generated using 60 particles and unchanged implementation settings.

This map shows that FastSLAM also closes the loop for this data, returning to previously traveled areas before losing accuracy in the southwest corner. The windows in this corner at the LIDAR height causes this problem, as the light transmits through the glass and returns a maximum range value. This results in an open area visible in the maps, but also results in a less certain measurement model. When the scan distance is a maximum, the model does not use this scan as the sensor does not observe any obstacles to construct particle weights from. Without the ability to properly weight the particles, none are correct as the vehicle takes the corner a second time.

The additional data covers many of the same survey points in a slightly different order. The primary difference is the number of repeated locations as the vehicle travels around the loop a second time. Table 9 compares the error at the surveyed points for the FV-SIFT, MINS, and FastSLAM paths.

Table 9. Additional Data Error

Map Location	Survey Point	FV-SIFT error (m)	MINS error (m)	FastSLAM error (m)
Start	50	0.0	0.0	0.0
NE	34	0.763	0.527	0.503
NW	44	1.609	2.312	0.747
SW	83	3.772	5.786	1.599
SE	64	4.030	5.964	0.861
NE	34	2.549	2.833	0.471
NW	44	2.550	5.028	0.779
SW	83	6.014	9.912	1.107
Finish	63	5.235	9.103	5.506
mean		2.947	4.607	1.286

As discussed, the FV-SIFT path is closer to truth than the MINS path for this data. This is partly due to Table 9 displaying only errors at specific points and not differences in heading. FastSLAM produces a much more accurate path than both when using MINS for input, decreasing MINS error by 72% to average 56% less error than FV-SIFT.

4.6 Summary

The FastSLAM in this research does not produce the ideal path, but is remarkably close and produces a much better map than any other presented. Most importantly, the map in Figure 20 is suitable for mission planning and accurate enough for local area navigation.

This FastSLAM map is achieved from the data gathered by a vehicle operating stereo cameras, an IMU, odometry, and a LIDAR. This vehicle is driven manually to gather all sensor data around an indoor environment consisting of a hallway loop.

The collection results in images and text files as input to the MINS and FastSLAM implementations.

The implementation runs in two phases. The first phase processes the stereo images and extracts an egomotion path. The second phase combines the egomotion path with an integrated IMU path and vehicle odometry in the MINS system. It then uses the MINS path and LIDAR ranges to execute FastSLAM and construct occupancy grid maps of the environment.

V. Conclusion & Future Work

When the environment is unknown, the SLAM problem states that a vehicle must construct a map detailing its surroundings while using the map it is building to maintain an accurate location. Such a vehicle is faced with the fundamentally difficult, and circularly defined SLAM problem. This research presents the first SLAM solution to integrate stereo cameras, an IMU, and odometry into a MINS path, then combines the MINS path with a LIDAR in a FastSLAM implementation. A P2-AT8 equipped with these sensors completed an indoor hallway loop generated real data for testing. The MINS path is more accurate than the FV-SIFT path for navigation, and the FastSLAM map is more accurate than an implementation using only the odometry and LIDAR.

Both the MINS and FastSLAM implementations presented in this document succeed in producing accurate maps, but remain a reach from being perfect. Accordingly, this chapter discusses the various positive and negative components in the implementation. Later, it covers several related topics that may be used in the future to improve on this implementation or in areas related to SLAM and navigation.

5.1 Conclusions

The MINS system presented in this research is effective at its purpose but is also complicated to compute. Creating and operating Kalman filters requires a certain area of expertise, separate from knowledge of robotics and mapping algorithms.

The Kalman filter implemented is remarkably simple compared to similar solutions such as FV-SIFT [23, 65]. The MINS filter could most likely be replaced to similar effect by a weighted average of inputs as this is the same process the filter carries out with different computational requirements.

The most time consuming and intensive aspect of this research surrounds the feature tracking and egomotion aspects of using cameras for navigation. Interestingly enough, the camera images are also the most difficult input to use and produce the least recognizable path. This cannot simply be a difficulty of working with cameras, as other research has presented egomotion more successfully [55, 56]. FV-SIFT accomplishes a mostly accurate path using just cameras and the IMU using either SIFT or SURF, whereas this egomotion has only been attempted using SIFT. Instead, there appears to be a rather significant improvement in the methods presented here that is yet to be discovered.

In stark comparison, the odometry and IMU sensors both involve very simple calculations to process compared to more complicated image methods. By its nature, odometry records data in the exact format desired for navigation. The IMU integration requires only a handful of values and operations in $O(1)$ constant time. Both are feasible for a real-time online navigation solution, even on a very slow machine.

More interesting than their convenience is that the odometry and IMU paths in this research ended up being more accurate than the egomotion path. Not only is the IMU used to collect the data a small consumer grade unit, but the vehicle skid steering is not the ideal odometry hardware. Other vehicles produce odometry that is approximately accurate as the MINS path presented here without the additional sensors or computation.

Accurate odometry provides a great improvement to the accuracy of a mapping solution, but constant indoor conditions are not always available. A major success of the research presented is the combination of inaccurate odometry that drifts and a small, inexpensive IMU to achieve a path much more accurate in position than IMU and more accurate heading than odometry. Based upon the sensor data, these two sensors may be functional without using the images for egomotion.

This gives rise to the possibility that the cameras may be too much of a burden in certain situations. If MINS can incorporate odometry and inertial measurements together, it would not require the stereo cameras or all its associated calculations. This arrangement may simplify the implementation, but it also removes one of the sensors that provides an additional observation to sense the environment. Removing this observation lowers the durability of the system by making it rely more heavily on less sensors.

Experimenting with less or different sensors would be a good area for future SLAM research to look into. Along these lines, the remainder of this chapter discusses this possibility and several others for continuing this area of SLAM research and vehicle navigation.

5.2 Future Work

There are two main goals when it comes to mobile navigation and SLAM solutions. One is to improve on the accuracy and consistency of solutions, and the other is to reduce the computation time. Suggestions for future SLAM research to improve one or the other logically divide themselves into one of these areas. The remainder of this chapter discusses the potential topics under these two research objectives.

5.2.1 Mapping for Accuracy.

To improve the accuracy of a SLAM solution, previous results lean toward use of a RBPF regardless of sensors or map representation. Not only has it produced the most accurate maps when compared to other strategies, but it allows the user to vary the number of particles M, adjusting the algorithm to any anticipated situation [61].

The most accurate results are those that leverage the most accurate sensors. Use of tuned odometry such as that be the PowerBot help, but a more important method is to incorporate a scan matching algorithm [28, 29]. Regardless of the control input path, scanner technology such as the LIDAR unit used is much more accurate than the compounding error from any odometry or inertial path. Adjusting the motion model by scan matching moves the particle distribution only to the most likely areas.

One way to achieve a similar effect would be to incorporate RBPF particle error back into the MINS filter. This only applies to this implementation with an initial sensor fusion filter and a following mapping filter, but would likely improve its particle distribution in the same manner as LIDAR scan matching. After the RBPF weights each of its particles, those with higher weights form a new mean pose $\bar{\mathbf{s}}_t$. This mean could replace or affect Kalman filter state $\mathbf{x}_t$ either before or after the RBPF resamples.

An alternative to a more standard odometry and range scan approach would consider different types of sensors. The previous section discussed the possibility of removing the cameras from the MINS system, leaving odometry and an IMU for control inputs that could be as effective as the solution presented here.

One might also consider using higher quality components to improve the raw data keeping a similar algorithm. A larger IMU can have order of magnitude improvements in measurement accuracy over the model used, so this advantage could certainly be leveraged in this implementation [65]. One could also take the opposite approach, using more cheap components to keep costs low. Using two inertial units on either side of vehicle allows translation and rotation to be interpreted between both readings.

A different sensor arrangement would transfer a range scanner to a new platform with cameras and an IMU, removing odometry as a sensor. The success of the image aided inertial system has already been demonstrated for navigation on airborne platforms [23, 65]. Incorporating a range scanner to a light aircraft brings SLAM capability to a highly mobile platform bringing the advantages of an airborne vehicle to the SLAM problem.

Much of the research using cameras for mapping seeks to bring the SLAM problem down in scale. Rather than have a massive vehicle plotting areas the size of university campuses, cameras seek to lower the cost and change the sensor type in potentially hazardous environments. Future research done with passive sensors allows vehicles to step away from range based solutions using expensive LIDAR scanners.

Cameras are passive sensors, those that do not project energy into the environment. Range scanners like radar, sonar, or LIDAR units do this by their operation. The effect of shooting so much energy has unpredictable consequences depending on what a vehicle encounters.

Testing environments generally consist of empty hallways or solid obstacles, but this is not the case for SLAM application environments. The repetitive bursts of light or sound can not only damage items around of the vehicle, but also affect the environment and leave evidence of intrusion. In hostile environments, active sensors reveal the presence vehicle as they are easily detected. An exploring vehicle equipped with passive sensors does not suffer this vulnerability.

5.2.2 Mapping for Speed.

If the main goal of a RBPF is to represent each possible pose a vehicle can have, the secondary goal of a RBPF implementation is to reduce the number of particles and still do this [46]. Reducing the number of particles in a filter enables FastSLAM to produce a map faster. Of course, dropping the number of particles down to 1 does not represent any distribution other than a single belief as presented [47].

A nice aspect of FastSLAM is that a more accurate belief affords a faster solution, as it requires less particles to cover a smaller distribution area. Researchers showed the effectiveness of scan matching by reducing the RBPF to 10 or 30 particles for a variety of different data sets [28, 29]. This allows them to compute the map much faster than if the same implementation used hundreds of particles. With either number of particles, the map may not be noticeably different. The challenging ground remains the balance of speed versus maintaining enough to produce an accurate map consistently.

For the stereo vision piece, there are several ways to improve the speed of calculation. One is to use a different feature tracking algorithm, as this contributes significantly to the image processing time. Using HOG for feature extraction instead of SIFT or SURF has been presented to take less time and achieve comparable results [14, 13]. Reducing feature extraction time would have a great affect on the time to obtain egomotion.

Another implementation could also limit the region of feature extraction using input from other sensors like the IMU [65]. This system also limits the number of features tracked to speed processing time. As an effect, the system only seeks matches to its tracked features and avoids unnecessary calculations. Implementation on a specific computer also speeds this process, something not accomplished in this research [23].

A common goal remains to implement a SLAM solution in real-time. Researchers have already produced maps whose computation time is less than the data collection [29]. However, developments have yet to present an actively running SLAM test, or market a vehicle featuring SLAM as part of its robotic capabilities.

Bibliography

[1] Armangué, Xavier, Helder Araújo, and Joaquim Salvi. "A review on egomotion by means of differential epipolar geometry applied to the movement of a mobile robot". *Pattern Recognition*, 36(12):2927–2944, 2003.

[2] Axe, David. "One in 50 Troops in Afghanistan Is a Robot". Wired Magazine Danger Room, February 2011.

[3] Bailey, Tim, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot. "Consistency of the EKF-SLAM Algorithm". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3562–3568. 2006.

[4] Bailey, Tim, Juan Nieto, and Eduardo Nebot. "Consistency of the FastSLAM Algorithm". *IEEE International Conference on Robotics and Automation*, 424–429. 2006.

[5] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". *European Conference on Computer Vision*, 404–417. 2006.

[6] Beeson, Patrick, Joseph Modayil, and Benjamin Kuipers. "Factoring the Mapping Problem: Mobile Robot Map-Building in the Hybrid Spatial Semantic Hierarchy". *International Journal of Robotics Research*, 29(4):428–459, 2010.

[7] Bierman, Gerald. *Factorization Methods for Discrete Sequential Estimation.* Academic Press, 1977.

[8] Blanco, Jose-Luis, Juan-Antonio Fernández-Madrigal, and Javier Gonzalez. "Towards a Unified Bayesian Approach to Hybrid Metric-Topological SLAM". *IEEE Transactions on Robotics*, 24(2):259–270, 2008.

[9] Borrmann, Dorit, Jan Elseberg, Kai Lingemann, Andreas Nüchter, and Joachim Hertzberg. "Globally Consistent 3D Mapping with Scan Matching". *Robotics and Autonomous Systems*, 56(2):130–142, 2008.

[10] Burgard, Wolfram, Cyrill Stachniss, Giorgio Grisetti, Bastian Steder, Rainer Kümmerle, Christian Dornhege, Michael Ruhnke, Alexander Kleiner, and Juan Tardós. "Trajectory-based Comparison of SLAM Algorithms". *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2009.

[11] Castellanos, José, José Neira, and Juan Tardós. "Limits to the Consistency of EKF-based SLAM". *International Federation of Automatic Control Symposium on Intelligent Autonomous Vehicles*, 1244–1249. 2004.

[12] Choset, Howie and Keiji Nagatani. "Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization". *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

[13] Dalal, Navneet. *Finding People in Images and Videos.* PhD Thesis, Institut National Polytechnique de Grenoble, 2006.

[14] Dalal, Navneet and Bill Triggs. "Histograms of Oriented Gradients for Human Detection". *International Conference on Computer Vision and Pattern Recognition*, 886–893. 2005.

[15] Dellaert, Frank, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "Monte Carlo Localization for Mobile Robots". *IEEE International Conference on Robotics and Automation*, volume 2, 1322–1328. 1999.

[16] Dissanayake, M.W.M. Gamini, Paul Newman, Steven Clark, Hugh Durrant-Whyte, and M. Csorba. "A Solution to the Simultaneous Localization and Map Building (SLAM) Problem". *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[17] Doucet, Arnaud, Nando de Freitas, Kevin Murphy, and Stuart Russell. "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks". *Conference on Uncertainty in Artificial Intelligence*, 176–183. 2000.

[18] Durrant-Whyte, Hugh and Tim Bailey. "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms". *IEEE Robotics and Automation Magazine*, 13(2):99–110, 2006.

[19] Ebcin, Sedat. *Tightly Integrating Optical And Inertial Sensors For Navigation Using The UKF.* MS Thesis, Air Force Institute of Technology, 2008.

[20] Eliazar, Austin and Ronald Parr. "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping without Predetermined Landmarks". *International Joint Conference on Artificial Intelligence*, 1135–1142. 2003.

[21] Eliazar, Austin and Ronald Parr. "Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps". *Advances in Neural Information Processing Systems*, 339–346. 2006.

[22] Elinas, Pantelis, Robert Sim, and James Little. "σSLAM: Stereo Vision SLAM Using the Rao-Blackwellised Particle Filter and a Novel Mixture Proposal Distribution". *International Conference on Robotics and Automation*, 1564–1570. 2006.

[23] Fletcher, Jordan. *Real-Time GPS-Alternative Navigation Using Commodity Hardware.* MS Thesis, Air Force Institute of Technology, 2007.

[24] Frese, Udo. "Closing a Million-landmarks Loop". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5032–5039. 2006.

[25] Frese, Udo. "Treemap: An O(log n) Algorithm for Indoor Simultaneous Localization and Mapping". *Autonomous Robots*, 21(2):103–122, 2006.

[26] Garcia, Miguel Angel and Agusti Solanas. "3D Simultaneous Localization and Modeling from Stereo Vision". *IEEE International Conference on Robotics and Automation*, 847–853. 2004.

[27] Gordon, Neil, D.J. Salmond, and A.F.M. Smith. "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation". *IEEE Proceedings on Radar and Signal Processing*, 107–113. 1993.

[28] Grisetti, Giorgio. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling". *IEEE International Conference on Robotics and Automation*, 2443–2448. 2005.

[29] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters". *IEEE Transactions on Robotics*, 23:34–46, 2007.

[30] Hähnel, Dirk, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. "An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 206–211. 2003.

[31] Hesch, Joel and Stergios Roumeliotis. "An Indoor Localization Aid for the Visually Impaired". *IEEE International Conference on Robotics and Automation*, 3545–3551. 2007.

[32] Hornung, Armin, Hauke Strasdat, Maren Bennewitz, and Wolfram Burgard. "Learning Efficient Policies for Vision-based Navigation". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4590–4595. 2009.

[33] Howard, Andrew. "Multi-Robot Simultaneous Localization and Mapping Using Particle Filters". *International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[34] Jones, Eagle and Stefano Soatto. "Visual-Inertial Navigation, Mapping and Localization: A Scalable Real-Time Causal Approach". *International Journal of Robotics Research*, 2010.

[35] Kaess, Michael and Frank Dellaert. "Probabilistic Structure Matching for Visual SLAM with a Multi-Camera Rig". *Computer Vision and Image Understanding*, 114(2):286–296, 2010.

[36] Kaess, Michael, Ananth Ranganathan, and Frank Dellaert. "iSAM: Fast incremental Smoothing and Mapping with Efficient Data Association". *IEEE International Conference on Robotics and Automation*, 1670–1677. 2007.

[37] Kahan, William. "Pracniques: Further Remarks on Reducing Truncation Errors". *Communications of the Association for Computing Machinery*, 8(1):40–48, 1965.

[38] Kootstra, Gert and Lambert Schomaker. "Using symmetrical regions of interest to improve visual SLAM". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 930–935. 2009.

[39] Leonard, John and Hugh Durrant-Whyte. "Simultaneous Map Building and Localisation for an Autonomous Mobile Robot". *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, 1442–1447. 1991.

[40] Lowe, David. "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision*, 2(60):91–110, 2004.

[41] Maimone, Mark, Yang Cheng, and Larry Matthies. "Two Years of Visual Odometry on the Mars Exploration Rovers". *Journal of Field Robotics, Special Issue on Space Robotics*, 24(3):169–186, 2007.

[42] Marks, Tim, Andrew Howard, Max Bajracharya, Garrison Cottrell, and Larry Matthies. "Gamma-SLAM: Using Stereo Vision and Variance Grid Maps for SLAM in Unstructured Environments". *Journal of Field Robotics*, 26(1):26–51, 2009.

[43] Maybeck, Peter. *Stochastic Models, Estimation, and Control*, volume 1 and 2. Academic Press, 1979.

[44] Milella, Annalisa and Roland Siegwart. "Stereo-Based Ego-Motion Estimation Using Pixel Tracking and Iterative Closest Point". *IEEE International Conference on Computer Vision Systems*, 21. 2006.

[45] Montemerlo, Michael. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* PhD Thesis, Carnegie Mellon University, 2003.

[46] Montemerlo, Michael, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". *AAAI National Conference on Artificial Intelligence*, 593–598. 2002.

[47] Montemerlo, Michael, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges". *International Joint Conference on Artificial Intelligence*, 1151–1156. 2003.

[48] Olson, Clark, Larry Matthies, Marcel Schoppers, and Mark Maimone. "Stereo Ego-motion Improvements for Robust Rover Navigation". *IEEE International Conference on Robotics and Automation*, 1099–1104. 2001.

[49] Olson, Clark, Larry Matthies, Marcel Schoppers, and Mark Maimone. "Rover Navigation Using Stereo Ego-Motion". *Robotics and Autonomous Systems*, 43(4):215–229, 2003.

[50] Olson, Edwin, John Leonard, and Seth Teller. “Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates”. *International Conference on Robotics and Automation*, 2262–2269. 2006.

[51] Paz, Lina Maria, Pedro Piniés, Juan Tardós, and Jose Neira. “6-DOF SLAM With Stereo-in-Hand”. *IEEE Transactions on Robotics*, 24(5):946–957, 2008.

[52] Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[53] Se, Stephen, David Lowe, and Jim Little. “Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks”. *International Journal of Robotics Research*, 21(8):735–758, 2002.

[54] Shi, Jianbo and Carlo Tomasi. “Good Features to Track”. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 593–600. 1994.

[55] Sim, Robert, Pantelis Elinas, and Matt Griffin. “Vision-based SLAM Using the Rao-Blackwellised Particle Filter”. *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 9–16. 2005.

[56] Sim, Robert, Pantelis Elinas, and James Little. “A Study of the Rao-Blackwellised Particle Filter for Efficient and Accurate Vision-Based SLAM”. *International Journal of Computer Vision*, 74(3):303–318, 2007.

[57] Smith, Randall, Matthew Self, and Peter Cheeseman. “Estimating Uncertain Spatial Relationships in Robotics”. *IEEE International Conference on Uncertainty in Artificial Intelligence*, 267–288. 1986.

[58] Spero, Dorian and Ray Jarvis. *A Review of Robotic SLAM*. Technical Report MECSE-4-2007, Monash University, 2007.

[59] Thomas, Stephen. *Real-time Stereo Visual SLAM*. MS Thesis, Heriot-Watt University, 2008.

[60] Thrun, Sebastian. *Robotic Mapping: A Survey*. Technical Report CMU-CS-02-111, Carnegie Mellon University, 2002.

[61] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[62] Thrun, Sebastian and Michael Montemerlo. “The GraphSLAM Algorithm with Applications to Large-scale Mapping of Urban Structures”. *International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[63] Titterton, David and John Weston. *Strapdown Inertial Navigation Technology*. The Institution of Electrical Engineers, 2nd edition, 2004.

[64] Venable, Don, Jacob Campbell, Jared Kresge, Daylond Hooper, Gilbert Peterson, and Michael Veth. *Performance Evaluation of Coupling Between Vehicle Guidance and Vision Aided Navigation*. Technical report, Air Force Institute of Technology, 2009.

[65] Veth, Michael. *Fusion of Image and Inertial Sensors for Navigation*. PhD Thesis, Air Force Institute of Technology, 2006.

Vita

2d Lt Christopher Weyers graduated in June 2005 from the Northfield Mount Hermon School in Massachusetts. He then entered undergraduate studies at Rensselaer Polytechnic Institute in Troy, New York, where he graduated Cum Laude with a Bachelor of Science degree in Computer & Systems Engineering. He was commissioned in May 2009 through AFROTC Detachment 550.

After attending the Air and Space Basic Course, he was assigned to Wright-Patterson AFB as a student in the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation in March 2011, he will be assigned to the Sensors Directorate, Air Force Research Laboratory, Wright-Patterson AFB, Ohio.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 24–03–2011	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sep 2009 — Mar 2011

4. TITLE AND SUBTITLE Multiple Integrated Navigation Sensors for Improved Occupancy Grid FastSLAM	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Weyers, Christopher P, 2d Lt, USAF	**5d. PROJECT NUMBER** ENG 11-194
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/11-08
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Jacob Campbell Air Force Research Laboratory, Sensors Directorate, RF Reference Systems 2241 Avionics Circle Wright-Patterson AFB, OH 45433 (937) 255-6127x4154, jacob.campbell@wpafb.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RYRN
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

An autonomous vehicle must accurately observe its location within the environment to interact with objects and accomplish its mission. When its environment is unknown, the vehicle must construct a map detailing its surroundings while using it to maintain an accurate location. Such a vehicle is faced with the circularly defined Simultaneous Localization and Mapping (SLAM) problem. However difficult, SLAM is a critical component of autonomous vehicle exploration with applications to search and rescue. To current knowledge, this research presents the first SLAM solution to integrate stereo cameras, inertial measurements, and vehicle odometry into a Multiple Integrated Navigation Sensor (MINS) path. The implementation combines the MINS path with LIDAR to observe and map the environment using the FastSLAM algorithm. In real-world tests, a mobile ground vehicle equipped with these sensors completed a 140 meter loop around indoor hallways. This SLAM solution produces a path that closes the loop and remains within 1 meter of truth, reducing the error 92% from an image-inertial navigation system and 79% from odometry FastSLAM.

15. SUBJECT TERMS

Bayesian filtering, mobile robots, machine vision, simultaneous localization and mapping, vehicle navigation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Gilbert L. Peterson (ENG)
U	U	U	UU	118	**19b. TELEPHONE NUMBER** *(include area code)* (937)255-3636x4281, gpeterson@afit.edu