2016

Localization And Mapping Of Unknown Locations And Tunnels With Unmanned Ground Vehicles

Doris Turnage
University of Mississippi

LOCALIZATION AND MAPPING OF UNKOWN LOCATIONS WITH UNMANNED GROUND VEHICLES

A Dissertation
presented in partial fulfillment or requirements
for the degree of Doctor of Philosophy
in the Department of Engineering Science
The University of Mississippi

by

DORIS M. TURNAGE

August 2016

ABSTRACT

The main goals of this research are to enhance a commercial off the shelf (COTS) software platform to support unmanned ground vehicles (UGVs) exploring the complex environment of tunnels, to test the platform within a simulation environment, and to validate the architecture through field testing.

Developing this platform enhances the U. S. Army Engineering Research and Development Center's (ERDC's) current capabilities and creates a safe and efficient autonomous vehicle to perform the following functions within tunnels: (1) localizing (e.g., position tracking) and mapping of its environment, (2) traversing varied terrains, (3) sensing the environment for objects of interest, and (4) increasing the level of autonomy of UGVs available at the ERDC.

The simulation experiments were performed in the STAGE Simulator, a physics-based multi-scale numerical test bed developed by Robotic Operating System (ROS). Physical testing was conducted in Vicksburg, MS using a Coroware Explorer. Both the simulation and physical testing evaluated three simultaneous localization and mapping (SLAM) algorithms, i.e., Hector SLAM, gMapping, and CORESLAM to determine the superior algorithm. The superior algorithm was then used to localize the robot to the environment and autonomously travel from a start location to a destination location.

Completion of this research has increased the ERDC's level of autonomy for UGVs from tether to tele-operated to autonomous.

DEDICATION

This work is dedicated to the memory of my parents, R. G. and Mary Turnage, my wonderful brother Elbert Turnage, my favorite cousin Hugh D. Brownridge, and family friends Eli Watson and Van Rideout. All of you provided encouragement, guidance, resources, and most of all love and laughter.

ACKNOWLEDGMENTS

I want to thank everyone for their help in editing the final document. I would also like to thank God for life.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

1.1. Background

Congress mandated that one-third of military vehicles be autonomous by 2015 (Kinney, et al., 2006). Autonomy, in this case, implies that the unmanned ground vehicle (UGV) should be able to traverse a dynamic and unstructured environment with little or no human intervention. In the broadest sense, a UGV is any piece of mechanized equipment that moves across the surface of the ground and serves as a means of carrying or transporting cargo, but explicitly does NOT carry a human being (Gage,1995). The military uses the UGV to perform reconnaissance and surveillance on the battlefield and in urban settings, tunnels, and other military missions. There exists a limited number of available robotic platforms and architectures equipped to navigate and perform mapping and localization of complex environments.

The goal of this research is to develop a software system to support UGVs exploration complex environments of tunnels and other subterranean areas such as basements, old mines, sewers, and caves. The Department of Defense (DoD) has used robotic platforms for various missions over the past 10 years. The Department of Homeland Security (DHS) uses robotic platforms in exploring drug tunnels along the southern U.S. border and Mexico (examples are given below). The development of a robust framework for autonomous UGVs will lead to an increase in the availability of robotic platforms and architectures to the DoD and DHS through the use of modeling and simulation.

September 11, 2001 was a very devastating day, one that forever changed the United States of America (USA). Terrorist attacks occurred at the World Trade Center (WTC) towers in New York City, New York, and The Pentagon in Washington, D.C. Tele-operated robots were used under the direction of the Center for Robot-Assisted Search and Rescue from September 11 to October 2, 2001 to search for victims and to help assess the structural integrity of the WTC foundation (R. R. Murphy, 2004). The robots were used for tasks that the rescuers or canines could not perform, for example, to either go into spaces too small for a human or to pass through an area still burning (R. R. Murphy, 2004). Before September 11, 2001, the Oklahoma City bombing on April 19, 1995 motivated an interest in the domain of rescue robotics for urban search and rescue (R. R. Murphy, 2004). Robin R. Murphy states that urban search and rescue (USAR) missions, which deal with man-made structures, have a different emphasis than traditional wilderness rescue or underwater recovery efforts and can be even more demanding on robot hardware and software design than military applications (R. R. Murphy, 2004).

During the search and rescue phase after September 11, 2001, the insertion of the Inuktum micro-VGTV robot into a sewer pipe at the WTC site allowed rescuers to attempt to locate an entry into a basement (R. R. Murphy, 2004). The robot was small in size and tethered, using a safety line for vertical entry (R. R. Murphy, 2004).

On March 20, 2003, the USA invaded Iraq, a war different from any other because of new technological advances. One such advancement was the Dragon Runner Reconnaissance Robot, developed by the National Robotics Engineering Centre (NREC). Weighing about 15 pounds, the Dragon Runner was designed to be light enough to toss into a window or up and down stairs (Voth, 2004). The U.S. Marine Corps used the Dragon Runner during Operation Iraqi Freedom with approximately a dozen deployed for sentry missions and urban

reconnaissance (Voth, 2004). When U.S. forces went into Iraq in 2003, they had a limited number of robotic units on the ground (Singer, 2008). By the end of 2004, the number was up to 150 (Singer, 2008). By the end of 2005, it was up to 2400, and it more than doubled the next year (Singer, 2008).

Since 1990, approximately 130 tunnels have been discovered along the border between San Diego, California, and Tijuana, Mexico, one of the more recent discoveries was on March 16, 2016. According to the *New York Daily News* article by Alfred Ng, federal agents apprehended a 415-yard tunnel between Mexicali, Mexico, and Calexico, California, after a 16-month investigation, seizing almost 3,000 pounds of marijuana. The tunnel had entrances at a restaurant in Mexico and a newly built house in California. From 2010 to date, the tunnels have become more sophisticated, spacious, and expensive. According to Alfred Ng, the March 16, 2016 tunnel is the first instance where drug traffickers bought property in the U.S. specifically for hiding a drug tunnel.

According to the Associated Press article by Elliot Spagat on November 27, 2010, the sophisticated cross-border tunnel, equipped with a rail system, ventilation, and fluorescent lighting, was the second discovery of a major underground drug passage in San Diego during November 2010. This tunnel was shut down by the U.S. Homeland Security and Mexican officials. The elaborate tunnel's length was 2,200 feet. The tunnel was between the kitchen of a home in Tijuana, Mexico, and two warehouses in San Diego's Otay Mesa industrial district. The cost of constructing these tunnels appears to be enormous because of the sophistication of their design. The first tunnel, discovered on November 3, 2010, spanned 600 yards and contained 25 tons of marijuana. Discovery of these tunnels has escalated in the past three years, along with border security efforts such as border fence installations, increased border agents, and the call for

DoD support from the DHS. The discovery of these tunnels greatly concern both to the DoD and the DHS. Both agencies have identified a requirement for methods that will easily explore and map out the unknown tunnel's environments without sending humans into the tunnels and endangering them. Developing robotic platforms to perform this task would mitigate the danger to humans.

1.2. Types of Unmanned Vehicles

The USA took a variety of vehicles to war; however, there were basically three distinct types of unmanned vehicles: (1) unmanned ground vehicles (UGV), (2) unmanned aerial vehicles (UAV), and (3) autonomous underwater vehicles (AUV). Each of these unmanned vehicles saw seen action in the Iraqi War, and some were used for tunnel exploration along the United States-Mexico border.

1.2.1. Unmanned Ground Vehicles

The U.S. Army uses two major types of autonomous and semi-autonomous ground vehicles: large vehicles and small vehicles. Examples of large vehicles are tanks, trucks, and high-mobility multiple-wheeled vehicles also known as the HUMVEEs. Examples of small vehicles, sized for carrying by a soldier in a backpack, are the PackBot and Talon. The PackBot and Talon move around on treads like small tanks.

Manufactured and sold by the iRobot Corporation of Burlington, MA, the PackBot weighs approximately 28 kg (61.78 pounds) and costs approximately $40,000. The PackBot performed reconnaissance and neutralization of unexploded ordnance (UXO) and improvised explosive device sites, which posed a major problem to the U.S. Military during the Iraqi War.

Manufactured and sold by Foster-Miller Inc., the Talon weighs approximately 38 kg (85 pounds) and costs approximately $60,000. The Talon performs a variety of functions from

reconnaissance to weapons delivery. Equipped with a robotic arm, uses of the Talon include surveillance or grabbing. The Talon performed search and recovery missions at the World Trade Center after the September 11 attack and various missions in the war in Afghanistan (Voth, 2004). Twenty Talon robots were deployed in Iraq in the beginning of 2003 and have accomplished approximately 10,000 or more missions (Voth, 2004).

1.2.2. Unmanned Aerial Vehicles

The UAV is an unmanned aerial vehicle, for example, a drone. The military has been the dominant customer for UAVs in the United States, spending nearly half a billion dollars annually on UAVs in recent years (Russell and Norvig, 1995). The organizations and tactical units within the Office of the Secretary of Defense, Army, Navy, Marine Corps, and Air Force are responsible for specifying, acquiring, and operating UAVs (Russell and Norvig, 1995). Examples of UAVs are drones and the MQ-9 Reaper.

1.2.3. Autonomous Underwater Vehicles

An AUV is an autonomous underwater vehicle. The main fields of AUV application include: (1) ocean exploration and monitoring of water medium, (2) marine geological survey, (3) inspection of the underwater engineering structures and pipelines, (4) search, inspection, and rescue operations, and (5) protection of environment and maricultures (Russell and Norvig, 1995). AUVs were used to search 3.5 million square meters of shallow water for mines in Operation Enduring Freedom in Iraq in 2003 (Edwards, et al, 2004). Examples of AUVs are the Autonomous Benthic Explorer and the Odyssey IV.

1.3. Proposed Research

Designed for relatively benign environments related to terrain, command and control, tethered robotic platforms such as pipe inspections systems, surveillance robots, explosive ordnance disposal robots, and mine clearance robots perform at a low level of autonomy (Doray et al., 2009). In contrast, a tunnel environment presents several challenges to a UGV: i.e., the complexity of the terrain, communications-limitations (GPS does not work underground), and limited amounts of preexisting data about the tunnel's environment.

This dissertation research focuses on building a software system to support UGVs in exploring tunnels. The goal of this research is to build a robust architecture to perform tunnel mapping and localization in an unknown environment, hence to increase the level of autonomy of a UGV. The current UGV designs cannot accomplish the functions such as localization and mapping to operate in tunnel environments while communicating with aboveground systems.

1.4. Organization of Report

Chapter 2 provides background information on current and past research on localization and mapping UGVs in varied environments. Chapter 3 provides information on the robotic platform used in this research, the software, and the selection of algorithms for localization and mapping in the prototype system. Chapter 4 provides details and the results of the simulation experiments using CoreSLAM, gMapping, and HectorSLAM. Chapter 5 provides details and the results of the physical testing experiments using CoreSLAM, gMapping, and HectorSLAM field testing for the robotic platform. Chapter 6 provides details on localizing the robotic platform to maps of various resolution generated by HectorSLAM, pathfinding with A*, path traversal with the Explorer localized to the generated map. Chapter 7 provides conclusions and future work.

CHAPTER 2

LITERATURE REVIEW

2.1. Introduction

This chapter provides an overview of research in the area of localization and mapping of robots in uncertain complex environments such as buildings, urban areas, mines, underwater, and tunnels. Also included is an overview of past and current research by the Engineering Research and Development Center (ERDC) on modeling and simulation and tunnel exploration.

2.2. Mapping

Robotic mapping addresses the problem of acquiring spatial models of physical environments through mobile robots (Thrun, 2002b). Tasks performed by the robot include the identification of features, such as landmarks, distinctive objects, or shapes and estimation of the robot's location in reference to the identified features. One of the fundamental tasks in robotics is the creation of a map of the area where the robot is moving (Rozman, 2009). The robot uses the created map for its navigation in this environment.

The robot mapping problem, more specifically, consists of the robot finding its pose, creating a map, and integrating the two. The robot has to keep track of its state based on data perceived from its sensors. The pose, the most important part of the state, provides information related to the location and orientation of the robot relative to its environment, i.e., pose = (x, y, Θ), where x and y represent the location and Θ is the orientation of the robot relative to a given coordinate frame.

2.2.1. Robotic Mapping Problems

Sebastian Thrun lists the five following problems associated with robotic mapping: (1) measured noise, (2) map size, (3) correspondence/data association problem, (4) dynamic environments, and (5) robotic exploration (Thrun, 2002b).

As discussed earlier, robots, equipped with varied sensors to perceive their environment, navigate their environment seeking landmarks. As the robot navigates its environment, the sensor data may contain errors in measuring its environment. The errors are also defined as noise. The cause of the errors may be slippage caused by odometry errors or sensor noises because of real world predicaments. Odometry errors will accumulate and throw off an entire map.

Second, there is the problem of the size of the map. The size of the map may be increased as the robot navigates. When mapping a robot's environment, the information about the environment has to be stored. Storing the map requires more memory space and computational time as the map increases in size.

The third and possibly hardest problem in robotic mapping is the correspondence problem, also known as the data association problem. The data association problem is associated with differentiating between sensor measurements taken at time t corresponding with the same physical object in the world. The correspondence problem determines if two data points taken from different scans are the same object.

The fourth problem, dynamic environment, deals with environmental change as time passes, e.g., a landmark that is a moving object such as a person. Changes in an environment can be from slow to fast.

The final problem is robotic exploration during mapping, the task of generating robot motion in the pursuit of building a map (Chui Ching Yee, 2008). Figure 1 depicts a map of an UGV while exploring a tunnel and the map that it creates during the exploration of the tunnel.

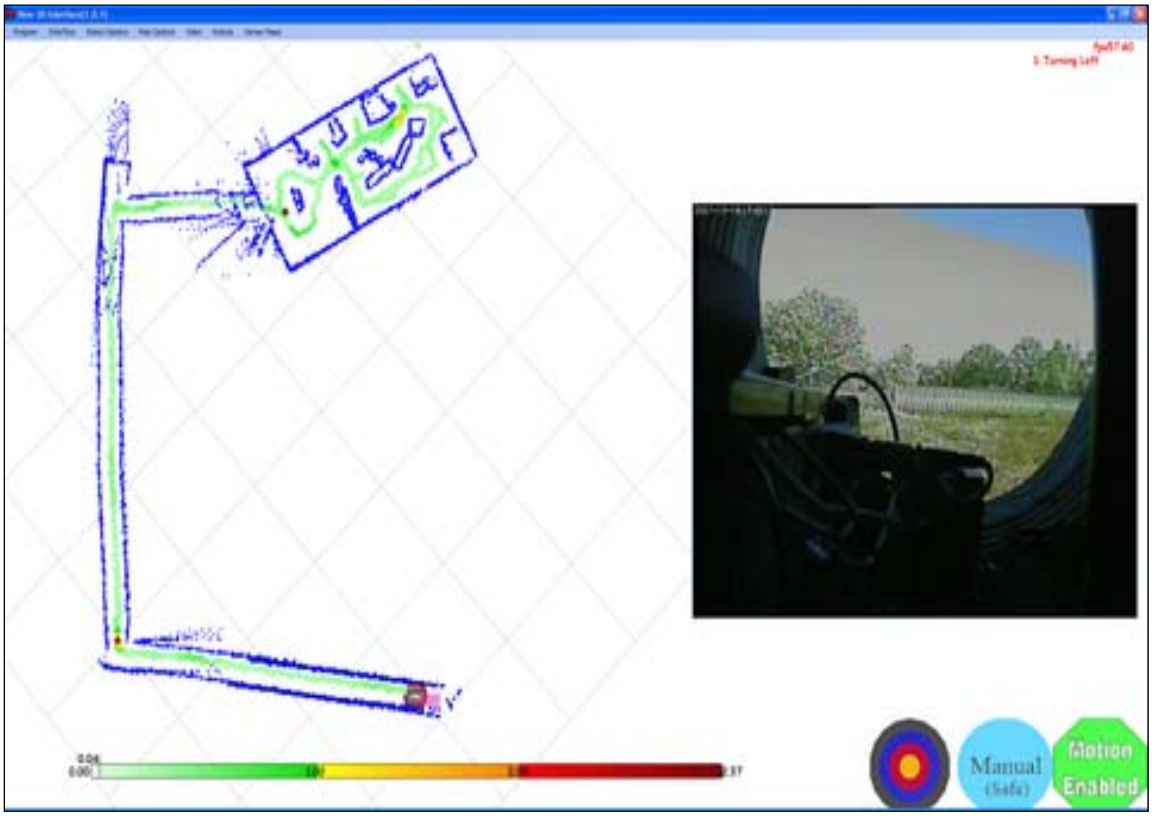

Figure 1. Map of area robot has navigated (Chui Ching Yee, 2008).

2.2.2. Metric and Topological maps

According to the type of maps generated, mapping algorithms can be roughly divided into two categories: metric approaches and topological approaches. The first approach models the environment using a metric map, enabling accurate estimation of the robot's positions. A metric approach typically provides a dense representation of the environment; therefore, it is well suited to precise trajectory planning (Angeli, et al., 2008). In a topological approach, the environment is segmented into distinctive pieces that form the nodes of a graph or topological map. The neighboring relations (i.e., whether or not a piece is accessible from another one) is

modeled using the edges of this graph (Angeli, et al., 2008). Topological mapping relies on a higher level of representation than metric mapping, allowing symbolic goal-driven planning and navigation. Compared with metric mapping, topological mapping usually provides a more compact representation that scales better with the size of the environment.

2.2.3. Probabilistic Mapping Algorithms

The correspondence problem is the key to solving either the metric mapping or topological mapping problem, i.e., the robot must be able to determine if data taken at different times corresponds to the same physical object (Reynolds, 2005). In this regard, probabilistic techniques yield some of the most accurate results of any of the methods.

At the foundation of any probabilistic algorithm for robotic mapping lies Bayes' Rule (1), where x represents the map and d represents the data for the sensors. $\Pr(x)$ is the prior probability of the map, and $\Pr(x|d)$ is the probability of the map x is true given the sensor measurement d, and $p(d|x)$ is the probability of the sensor measurement being d given an object at x.

$$\Pr(x|d) = \frac{p(d \mid x)\Pr(x)}{p(d)} \quad (1)$$

Usually, a Bayes estimator approximates both the map and the robot's pose.

The correspondence problem also can be tackled in an incremental fashion. For example, Reynolds (2005) discussed a maximum likelihood approach that compares nearby measurements of the previous map to identify the path the robot has moved within a small time frame. This approach was shown to be robust in the sense that it can recover from a wrong correspondence. Nevertheless, it takes a large amount of processing, making it less ideal for real-time applications.

An occupancy grid is one of the most popular incremental algorithms because of its ease of use and robustness. In the most basic form, the binary occupancy of a location (x,y) is

calculated and the cells (grids) are incrementally updated (Reynolds, 2005). This approach works well in a real-time application; however, it relies heavily on odometry data. This means that the errors in odometry data will accumulate, and hence, skew the map.

The above error accumulation problem can be alleviated by a hybrid method, e.g., combining a Bayes estimator with an incremental algorithm. Although they are typically more difficult to implement, they provide significantly better results than either a purely Bayes estimator or an incremental method for autonomous robots (Reynolds, 2005).

2.3. Localization

Localization refers to the estimation of the position of a mobile robot on a known or a predicted map (Stanculescu and Sojka, 2008). It requires using a map to interpret sensor data to determine the configuration of the robot. Without the ability to localize itself in an environment successfully, a robot is effectively stripped of its ability to do useful work (Kramer, 2010).

There are three issues involved with localization: the local position tracking problem, the global position tracking problem, and the kidnapped robot problem. Each problem has been extensively researched in the literature.

The local position tracking problem has received the most attention. When dealing with this problem, the robots initial pose is known. For global position tracking, the robot is unaware of its initial pose and has to determine its location from scratch. In the kidnapped robot problem, the robot knows its location; however, it is relocated to another location and is totally unaware of its new location. The mobile robot must figure out its location on its own completely. This problem is used to test a robot's ability to localize itself.

Although there are many methods for determining the location of the robot, in this research three methods are discussed in detail as follows: (1) Monte Carlo Localization,

(2) Kalman Filters, and (3) Markov Localization. Next, we provide a summary of which of the three localization problems these localization methods solve, along with one other possible solution to both the localization and mapping problems.

2.3.1. Monte Carlo Localization

Monte Carlo Localization (MCL) has been widely used to estimate a robot's pose and to solve the global localization problem, in which the robot does not know its starting position (Thrun,et al., 2001). MCL consists of four steps: sampling, prediction, update, and weight normalization as depicted in Figure 2 below. The general idea is as follows.

1. Initialize a set of samples (the current samples) so that their locations are evenly distributed and their importance weights are equal.
2. Repeat until done (i.e., weights converge) with the current set of samples:
    i. Move the robot a fixed distance and take a sensor reading.
    ii. Update the location of each of the samples (using the movement model).
    iii. Assign the importance weights of each sample to the likelihood of that sensor reading given that new location (using the sensor model).
    iv. Create a new collection of samples by sampling (with replacement) from the current set of samples based on their importance weights.
    v. Let this new collection be the current set of samples.

Research performed by Stanculescu and Sojka (2008) evaluates the Monte Carlo Localization algorithm. It was concluded that MCL can efficiently estimate the position of a robot on a grid-based map.

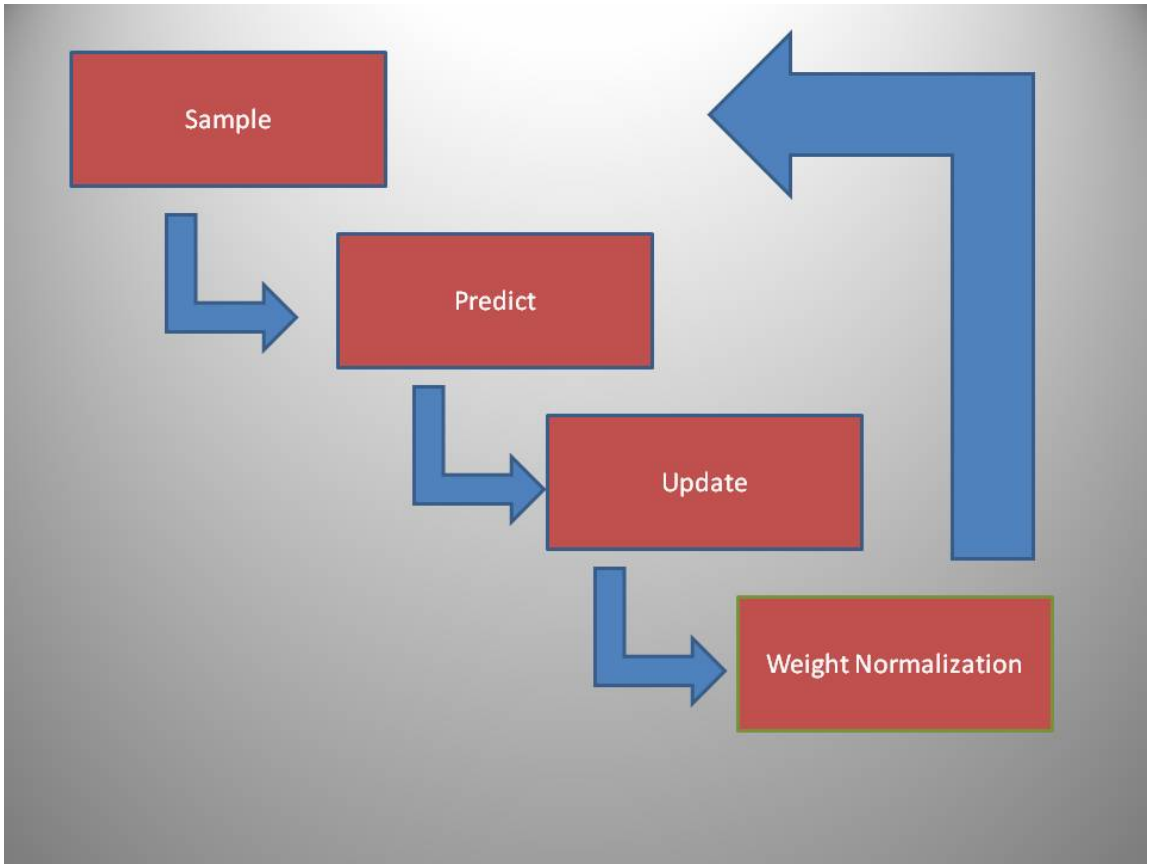

Figure 2. Monte Carlo Localization Algorithms.

In (Dellaert, al. , 1999), the researchers introduce a MCL method that represents the probability density involved by maintaining a set of randomly drawn samples. By using a sampling-based representation, they obtain a localization method capable of representing an arbitrary distribution. Experimentally, they show that the resulting method can efficiently localize a mobile robot without knowledge of its starting position, which is faster, more accurate, and less memory-intensive than earlier grid-based methods.

One advantage of using the MCL is its ability to represent multi-modal distributions; hence, it can be used to globally localize a mobile robot. A second advantage is that MCL drastically reduces the amount of memory required compared to a grid-based Markov localization (discussed in Section 2.3.3) and can integrate measurements at a considerably higher

frequency. A third advantage is that like many randomized algorithms, it is easy to implement (Stanculescu and Sojka, 2008).

Many variants of the MCL method exist, one such method is Mixture – Monte Carlo Localization, a mobile robot localization algorithm. It is a version of particle filtering that combines a regular sampler with its dual (Thrun, et al., 2001). The Mixture-MCL provides efficiency, versatility, resource adaptiveness, and robustness.

2.3.2. Kalman Filters

In a seminal paper, R. E. Kalman (1960) developed a probabilistic filtering algorithm for a control system. The Kalman Filter has been widely applied in robot perception. Its purpose is to use sensor measurements observed over time, containing noise (random variation) and other inaccuracies, to produce values that tend to be closer to the true values of the measurements than the observed values.

Kalman filters are essentially Bayes' filters under a Gaussian assumption. They have been widely applied to estimate the robots pose. The Kalman filter algorithm operates on the postulation that the current state function must be a linear step from the previous state function with Gaussian noise. The Kalman Filter is an effective approach in several aspects: (1) it supports estimations of past, present, and even future states, and (2) it can do so even when the precise nature of the modeled system is unknown (Welch and Bishop, 1995).

Although Kalman-filterbased techniques have proven to be robust and accurate for keeping track of the robot's position, it relies on the Gaussian assumption and lacks the ability to globally (re-)localize the robot in the case of localization failures (Dellaert, et al., 1999). Although the Kalman filter can be amended in various ways to cope with some of these difficulties, recent approaches have used richer schemes to represent uncertainty, moving away

from the restricted Gaussian density assumption inherent in the Kalman filter (Dellaert, et al., 1999).

2.3.3. Markov Localization

Markov localization is a special case of probabilistic state estimation using Bayes' rule. It is robust against both inaccurate maps and noisy sensors (Song, 2002). In many probabilistic mobile robot localization literatures, the term Markov Localization and Bayesian estimation (filter) are used interchangeably. Markov localization uses a probabilistic framework to maintain a position probability density over the whole set of possible robot poses. It is based on the Markov Assumption, i.e., the environment is static (Fox, et al. 1999). Instead of maintaining one hypothesis as to where in the world a robot may be, it maintains a probability distribution over the space of all such hypotheses (Fox, et al., 1999). The probabilistic representation allows it to weigh these different hypotheses in a mathematically sound way (Fox, et al., 1999). The pseudo code of a general Markov localization algorithm is given in Figure 3.

Markov localization has produced excellent success in mobile research areas, such as working as guidance robots at a crowded museum. Minerva and Rhino are two mobile robots using different types of robots and sensor modalities to act as interactive museum tour-guides. The major advantage of Markov localization is its ability to detect localization failures and to re-localize the robot (Fox, et al., 1999). The only disadvantage lies in the fixed representation of the grid, which has an undesirable effect; the memory requirement stays constant even if only a minor part of state space is updated.

2.4. Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is a technique used by robots and autonomous vehicles to build a map within an unknown environment (without apriori

knowledge) or to update a map within a known environment (with apriori knowledge from a given map) while keeping track of their current location. SLAM was introduced by R. Smith, M. Self, and P. Cheeseman in 1990, which laid the groundwork for the modern SLAM problem using Extended Kalman Filters (Smith, Self, and Cheesman, 1990).

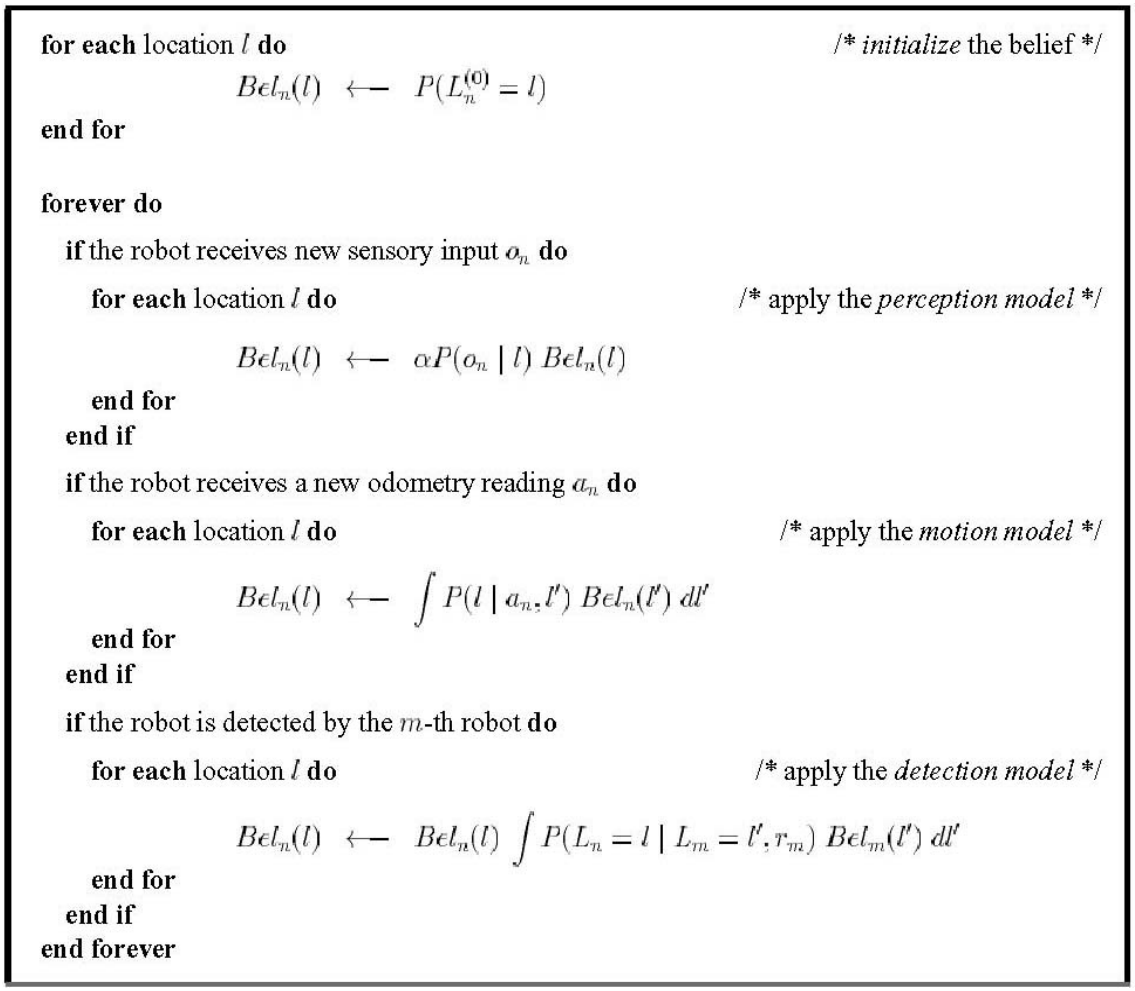

Figure 3. Markov Localization Algorithm (Fox, et al., 1999).

A solution to the SLAM problem has been seen as the "Holy Grail" and would enable robots to operate in an environment without a priori knowledge of obstacle locations. The characteristics of SLAM couple the problems of localization and mapping. The two quantities are to be inferred from a single measurement. Loosely speaking, a SLAM process consists of

multiple steps: landmark extraction, data association, state estimation, state update, and landmark update (Riisgaard and Blas, 2004).

To achieve SLAM, two models are required, the process model and the observation model. In the process model, a vehicle traverses through an environment containing a population of landmarks with a known kinetic model. In the observation model, the vehicle is equipped with a sensor that can take measurements of the relative location between any individual landmark and the vehicle itself. Various versions of SLAM exist: FastSLAM 2.0, Marginal SLAM, and EKF –SLAM, to name a few.

2.5. Research Goal

Currently, a very limited number of robotic platforms are available for tunnel exploration. The goal of this research is to develop a software platform to support a UGV to perform localization and mapping of a tunnel. The robotic platform should be able to detect humans and objects while exploring the tunnel. This research aids in expanding the number of robotic platforms available for use by the DoD and the DHS. Also, another goal is to increase the level of autonomy for UGVs in which the ERDC currently use from tethered UGV. Next, we discuss the modeling and simulation, tunnel exploration, and levels of autonomy.

2.5.1. Modeling and Simulation

Modeling and simulation (M&S) is a tool that saves time, money, and lives. M&S has been used by the military since the early 1940s. According to the Department of Defense's Modeling and Simulation Primer, military analysts use M&S to help shape the size, composition, and structure of forces to meet national military requirements, and to assess the sufficiency of operational plans. The Primer also states that the military acquisition community uses M&S (1) to evaluate requirements for new systems and equipment; (2) to conduct research,

development, and analysis activities, (3) to develop digitized prototypes and avoid the building of costly full scale mockups, and (4) to plan for efficient production and sustainment of the new systems and equipment when employed in the field.

The Department of Defense's Modeling and Simulation Primer defines a model as a physical, mathematical, or logical representation of a system, entity, phenomenon, or process with no movement, as in a replica of a car or airplane. The model may take the form of a set of assumptions concerning the operation of the system (Banks and Carson, 1984). Once developed and validated, a model can be used to investigate a wide variety of "what if" questions about the real world system.

In Discrete-Event System Simulation, simulation is defined by Banks as the imitation of the operation of a real-world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operation of characteristics of the real system (Banks and Carson, 1984).

2.5.2. Tunnel Exploration

The Unmanned Tunnel Exploration (UTE) research effort's goal is to develop a Semi-Autonomous Unmanned System for Small Unmanned Ground Vehicles in tunnel environments, and Intelligence, Surveillance, and Reconnaissance (ISR) processes (Doray, et al., 2009). The UTE research effort combines high resolution vehicle simulations with field experiments to advance the operational capability of SUGV's operating in tunnels. The UTE technology will have a significant impact on successfully deploying the first generation Tunnel Activity Detection Systems (TADS) solution through the Joint Task Force-North (JTF-N), a subordinate command of US NORAD-NORTHCOM, and the US ARMY ERDC. Further collaboration is

being conducted with TAE Technical Support Working Group, US ARMY TRADOC Analysis Center, Monterey, and Idaho National Labs. Figure 4 depicts the TALON navigating a tunnel in San Diego.

Figure 4. Tunnel exploration in San Diego, California.

2.5.3. Levels of Autonomy

An autonomous navigation system or autonomous robot can function, operate, or make decisions independently under reasonable circumstances, with limited human intervention. Increasing levels of human intervention decrease the autonomy of the navigation system. A classification of levels of autonomy takes into account the interaction between human control and the machine motions: teleoperation, supervisory, task-level autonomy, and full autonomy. In teleoperation, a human controls each movement. Each machine actuator change is specified by

the operator. At the supervisory level, a human specifies general moves or position changes, and the machine decides specific movements of its actuators. At the task-level autonomy, the operator specifies only the task, and the robot manages itself to complete it. In full autonomy, the machine will create and complete all its tasks without human interaction. Generally, higher levels of autonomy do not necessarily require more complex cognitive capabilities. For example, robots in assembly plants are completely autonomous, but operate in a fixed pattern.

The Autonomy Levels of Unmanned Systems (ALFUS) working group's definition of levels of autonomy is displayed in Figure 5. The ALFUS level ranges from zero to 10, where at zero there is no autonomy and the robot is controlled by remote control and at 10 the system is a fully intelligent system with no human interface.

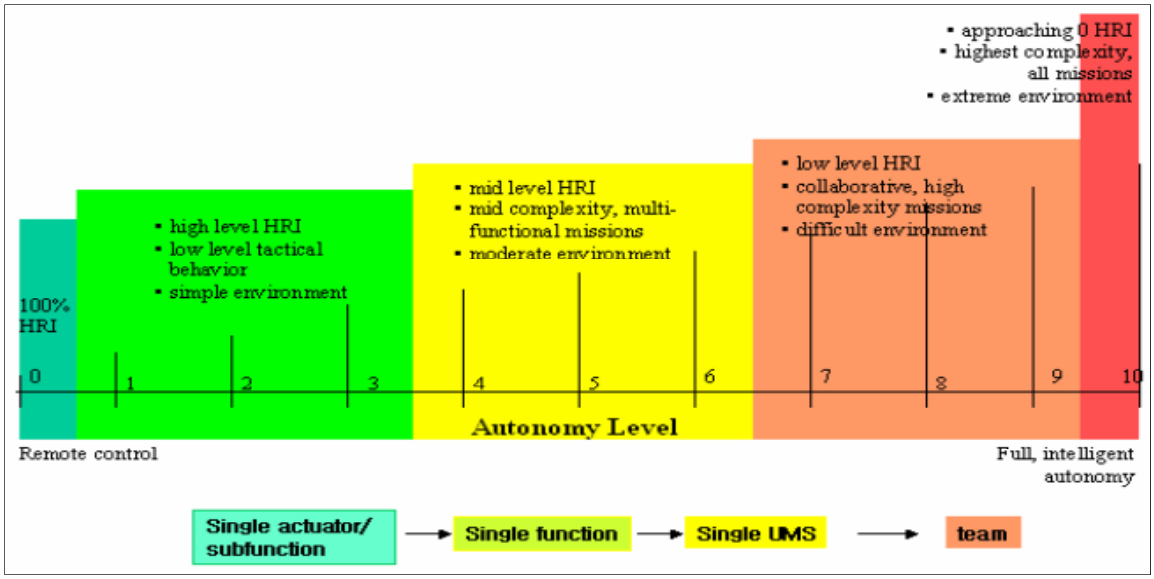

Figure 5. ALFUS defined Level of Autonomy.

2.6. Proposed Research

In this research, software was designed and tested for a UGV, the Coroware Explorer, to enable it to function as an UGV with the capability to localize and map an unknown area while navigating. The UGV was equipped with a camera, acoustic sensor, and a Laser Range Finder

that provides the UGV with readings to determine its pose and landmarks within the unknown area to create a map of the unknown research.

The developed system was validated using simulation and field testing to determine its ability to navigate the unknown environment, detect landmarks, and build a map of the navigated environment. Three scenarios were used for simulation testing. The simulated test environment was the STAGE simulator and the test sites were in facilities at the ERDC in Vicksburg, MS.

This research uniquely combines Coroware Explorer and field testing. In the next chapter, we introduce Coroware Explorer, the robot platform in the project, is introduced.

CHAPTER 3

OVERVIEW OF SYSTEM

3.1. Description of Robot Platform

The following UGV platforms received consideration as possible platforms for this research: (1) the Adept Pioneer 3-AT, (2) the Coroware Explorer, and (3) Superdroid Robots HD2 Treaded ATR Tank Robot Kit. Each of these platforms is a programmable robot, easily adaptable to varied sensors. Figure 6 depicts the Superdroid Robots HD2 Treaded ATR Tank Robot Kit, Figure 7 depicts the Coroware Explorer, and Figure 8 depicts the Adept Pioneer 3-AT.

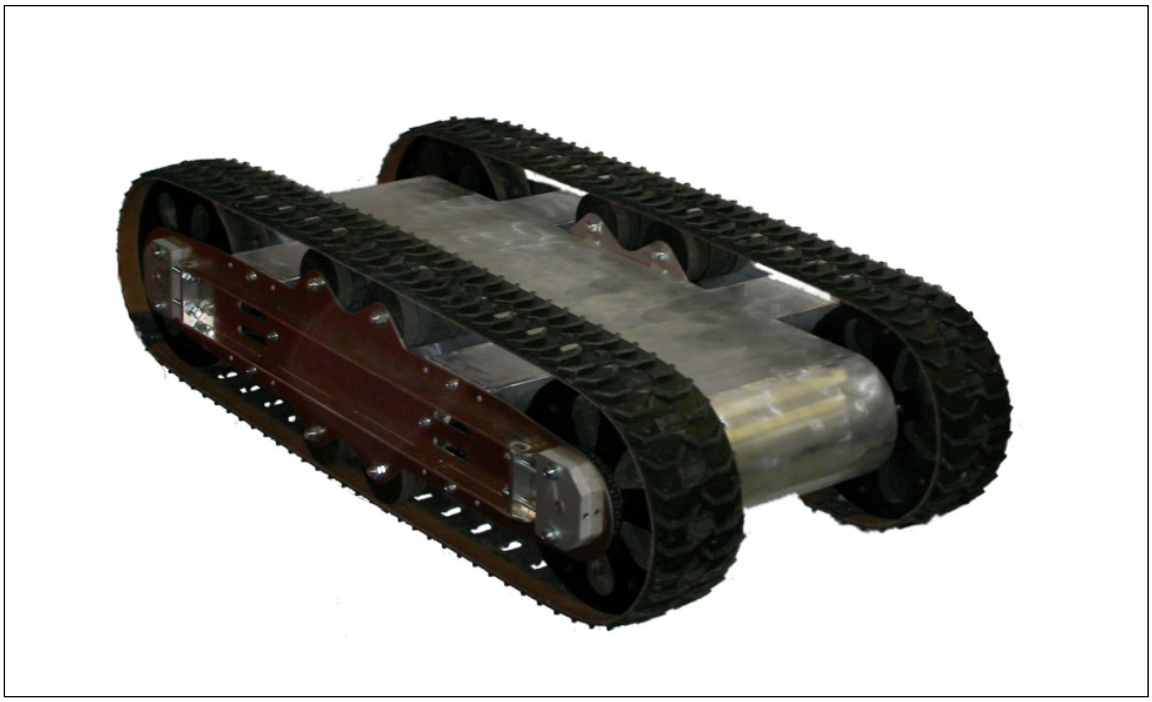

Figure 6. Superdroid Robots HD2 Treaded ATR Tank Robot Kit.

Figure 7. Coroware Explorer.

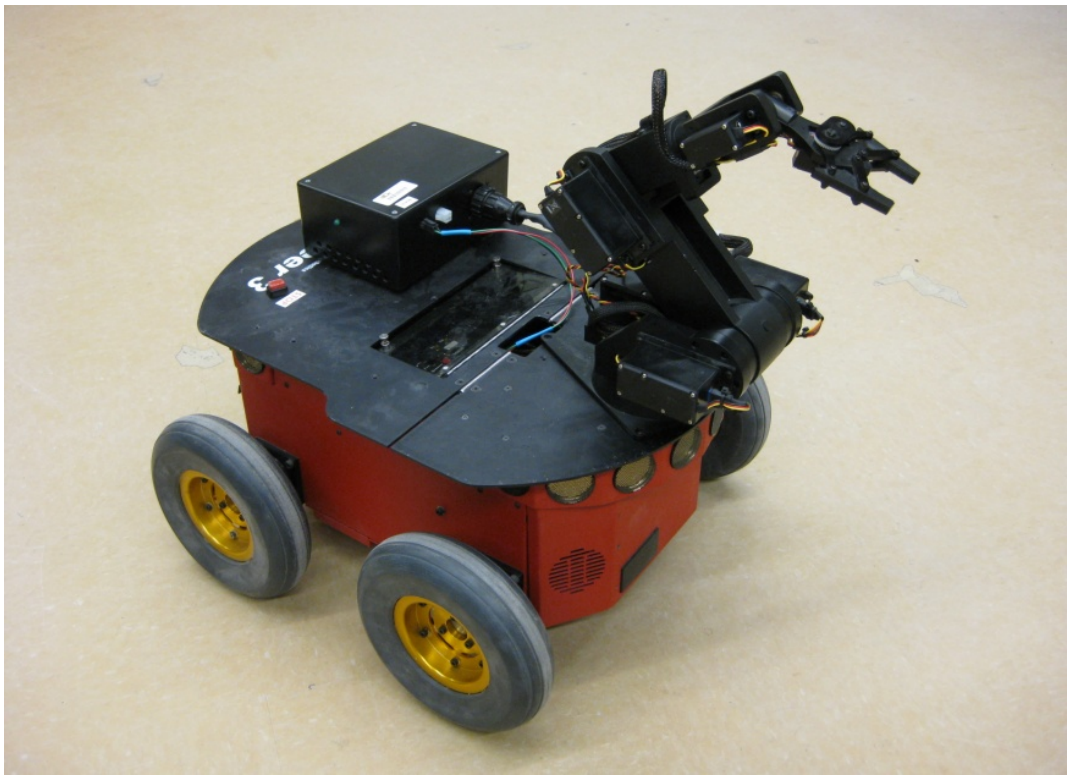

Figure 8. Adept Pioneer 3-AT.

The selection of the robot is based on several requirements.

- It may be tracked or wheeled.
- It must be capable of crossing objects of various sizes such as rocks, water, and gravel.
- It must support a range of sensors.

For this project, we selected the Coroware Explorer (Figure 7). The Explorer meets all the above requirements. It has six-inch clearance or more. It is equipped with two sensors: a two Mega Pixel Color Webcam to capture scenes and a Laser Range Finder (Figure 9) for detection of landmarks. It has a 2.0 GHz CPU, 1 GB of RAM, a 2.0 GHz CPU, a 4-hour battery life, a dual-boot operating system (Ubuntu Linux or Windows), Wi-Fi, and a CUDA-capable main board. Its dimensions are 23 inches long, 21 inches wide and 16 inches high, and with a weight of about 20 pounds. The robotic software platform is the Robotic Operating System. Appendix A contains an original quote for the Coroware Explorer from Coroware.

The Laser Range Finder, shown in Figure 9, is the Hokuyo UTM-30L-X. The Hokuyo UTM-30L-X has a detectable range of 100 mm to 30,000 mm, 25 milliseconds per scan, 12V operating voltage, and a 270 degree scanning range. It connects to the Explorer via USB. The cost of the Hokuyo UTM-30L-X was approximately $7000.00.

3.2. Description of Software

The Robotic Operating System (ROS), an Open Source UGV software, was selected for use with this project. ROS provides libraries and tools to help software developers create robot applications. ROS is not an operating system in the traditional sense of process management and scheduling; rather it provides a structured communication layer above the host operating system of a heterogeneous compute cluster (Quigley, et al., 2009).

Figure 9. Hokuyo Laser Range Finder.

ROS was designed to meet a specific set of challenges encountered when developing large-scale service robots as part of the STAIR project at Stanford University and the Personal Robots Program at Willow Garage, but the resulting architecture is far more general than service-robot and mobile-manipulation domains. The philosophical goals of ROS can be summarized as (1) peer-to peer, (2) tools-based, (3) multi-lingual, (4) thin, and (5) free and open-source.

ROS provides services expected of an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management (http://wiki.ros.org/ROS/Introduction). It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers (Quigley, et al., 2009).

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, although the ROS community has been contributing support for Fedora, Gentoo, Arch Linux, and other Linux platforms (Quigley, et al., 2009).

3.3. Mapping and Localization Algorithms

Thus far, grid-based, topological, and hybrid mapping algorithms were studied for possible use to perform this research. Markov Localization, the Monte Carlo Method, and Kalman filters have been studied as a means to perform localization of the robot. SLAM has been studied and performs localization and mapping simultaneously.

3.4. Summary

The Coroware Explorer was selected as the platform for this research. ROS was used to program the Explorer robotic platform. Three mapping and localization algorithms at will be implemented and compared in Chapter 4.

CHAPTER 4

SIMULATION TESTING EVALUATION

This chapter provides a quantitative evaluation of three laser-based SLAM algorithms implemented in the 2-D simulator: CoreSLAM, Gmapping, and HectorSLAM. The tele-operated robot explores three ground truth map images, and each produces a generated map image. Image registration, the process of aligning two images of the same scene, is then used to align the ground truth map and the generated map for a multimodal comparison of the two. The Hausdorff Distance, a mathematical function used to measure the difference between two subsets of the same space, is then calculated to show the difference in the two images as a numeric value. Both image registration and the Hausdorff distance code are written in MATLAB and produce results comparing each of the algorithms and showing one algorithm is superior to the other two.

4.1. Evaluated SLAM Algorithms

As discussed in Section 2.4, many types of SLAM algorithms exist, for example, those that are vision-based or laser-based, and those that are 2-D or 3-D. The three algorithms evaluated were available at www.ros.org: CoreSLAM, Gmapping, and HectorSLAM. Each algorithm requires laser-based inputs as data for the simulation. The three SLAM algorithms collect data via the tele-operated robot within the Stage 4.1.1 software simulation, each outputting a map image of the navigated environment.

Each algorithm is used as a black box in this research; however, the underlying details of the implementation differ in the following aspects: (1) Both HectorSLAM and CoreSLAM rely

on scan matching, while Gmapping uses particle filters, (2) CoreSLAM may produce a different map each time with the same input dataset, and (3) CoreSLAM requires loop closing while Hector SLAM does not.

4.1.1. CoreSLAM

CoreSLAM is a version of SLAM that implements tinySlam. It requires a mobile robot that provides odometry data, and is equipped with a horizontally mounted, fixed, laser range-finder. The slam_CoreSLAM node will attempt to transform each incoming scan into the Odom (odometry) “tf” frame.

CoreSLAM relies on a simple Monte Carlo algorithm for scan matching and was developed by Steux and El Hamzaoui with the goal of producing a SLAM algorithm with no more the 200 lines of codes. CoreSLAM has a particle filter routine, ts_distance_scan_to_map, and a map update function. The ts_distance_scan_to_map routine tests each state position, and the map update function updates the map as the robot navigates its environment.

The slam_CoreSLAM node takes as input laser data and pose data collected from the laser range finder and outputs a low quality map, yet a recognizable one.

Overall, CoreSLAM performs better on a slow robot.

4.1.2. Gmapping

Gmapping is a highly efficient Rao-Blackwellized particle filter to learn grid maps from laser range data (Stahniss, et al., 2011). Implementation requires a mobile robot equipped with a mounted, fixed, laser range finder.

Loop closure is the hardest part; when closing a loop, be sure to drive another 5 to 10 meters to get plenty of overlap between the start and end of the loop.

This package contains Gmapping from OpenSlam and a ROS wrapper. The Gmapping package provides laser-based SLAM as a ROS node called slam_Gmapping. Using slam_Gmapping, it creates a 2-D occupancy grid map (like a building floor plan) from laser range finder and pose data collected by a mobile robot (Santos, n.d.).

The slam_Gmapping node takes as input laser data and pose data collected from the laser range finder and outputs a high quality map.

4.1.3. Hector SLAM Gmapping

Hector SLAM relies on scan matching, uses a Gauss-Newton Approach, and is accurate enough that it doesn't require loop closure. The Hector SLAM package consists of three main packages, hector_mapping, hector_geotiff, and hector_trajectory_server.

The Hector_mapping node is a SLAM approach used with or without odometry on platforms that exhibit roll/pitch motion (of the sensor, the platform, or both). It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2-D pose estimates at the scan rate of the sensors (40 Hz for the UTM-30LX). Although the system does not provide an explicit loop closing ability, it is sufficiently accurate for many real-world scenarios. The system has been used successfully on Unmanned Ground Robots, Unmanned Surface Vehicles, Handheld Mapping Devices, and logged data from quadrotor UAVs (Kohlbrecher, n.d.).

Hector_geotiff saves the map and robot trajectory to geotiff image files. The hector_trajectory_server saves trajectory files as output. The hector_mapping node's main input is scan data on the /scan topic. The data are then transformed via the /tf topic.

Overall, Hector SLAM outputs a high quality map that is recognizable.

4.2. Ground Truth Maps

Figure 10 displays the three maps chosen as the ground truth maps for the simulation, labeled as Map One, Map Two, and Map Three. Each is simple in design and simple to navigate. All three maps were downloaded from the web and used with the Stage 4.1.1 simulator as the ground truth map for the simulated robot to navigate.

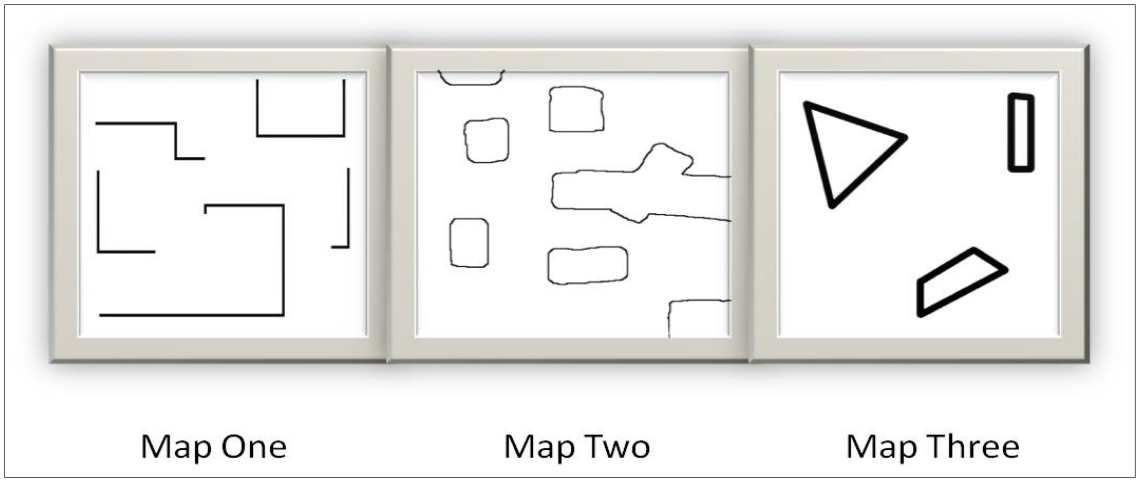

Figure 10. Ground Truth Maps.

4.3. Simulation Results

Each of the three SLAM algorithms previously discussed was tested using the 2-D simulation environment, Stage, a 2-D robotic simulator that provides users with the capabilities of simulating a robot or a variety of robots in an environment or a variety of environments. Stage, an open-source software, provides multiple physics-based models for robot sensors and actuators. Some of the currently supported models are sonar and infrared rangers, 2-D scanning laser rangefinder, color-blob tracking, fiducial tracking, bumpers, grippers, and mobile robot bases with odometric and global localization (University of Tennessee Knoxville, 2007).

One advantage Player/Stage provides is the ability to move from simulation to the robot by changing a few parameters (Staranowicz and Mariottini. 2011). The learning curve on the Stage software is a disadvantage.

Stage, used standalone or with ROS, has many versions. This research implements Stage 4.1.1, the most recent version and requires ROS Fuerte for implementation.

The simulated robot completely navigated each of the three ground truth maps with each of the three algorithms to produce the three generated maps. Figure 11 shows the ground truth map, labeled Fixed, and the three generated maps produced by each algorithm, labeled CoreSLAM, Gmapping, and Hector Slam, after navigating the ground truth of Map One. Figure 12 shows the ground truth map, labeled Fixed, and the three generated maps produced by each algorithm, labeled CoreSLAM, Gmapping, and Hector Slam, after navigating the ground truth of Map Two. Figure 13 shows the ground truth map, labeled Fixed, and the three generated maps produced by each algorithm, labeled CoreSLAM, Gmapping, and Hector Slam, after navigating the ground truth of Map Three.

Analyses of these maps will be discussed later.

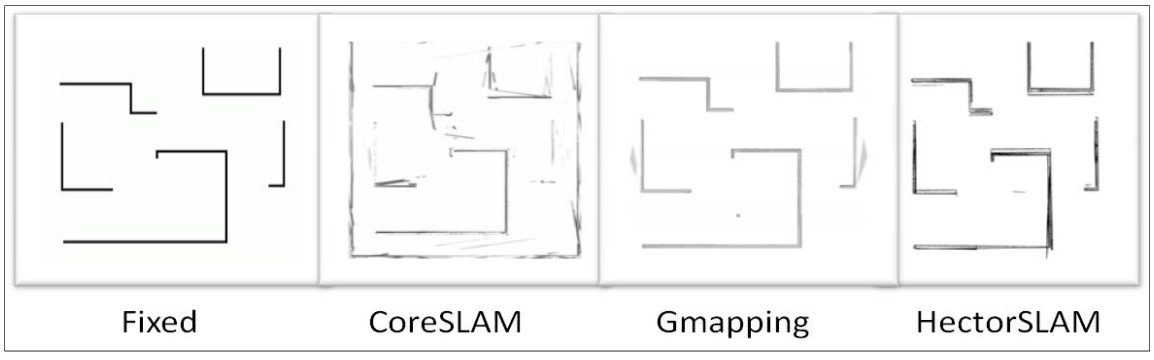

Figure 11. Map One and Generated Map of Each Algorithm.

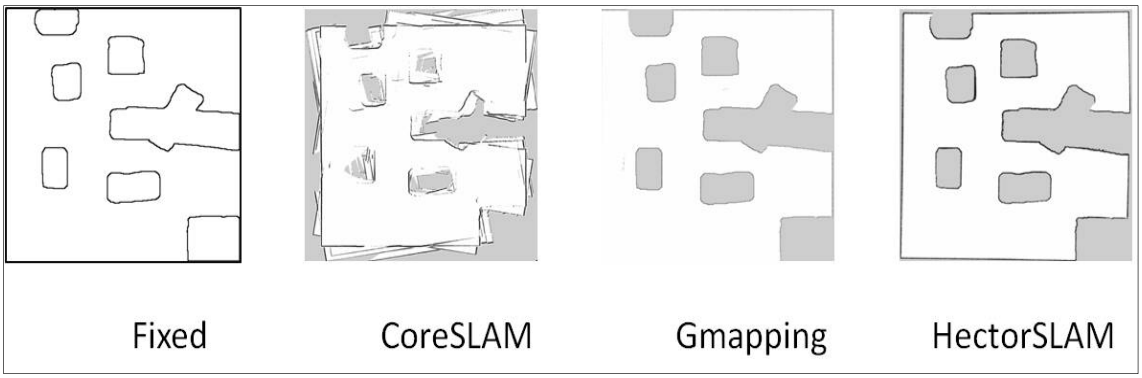

Figure 12. Map Two and Generated Map of Each Algorithm.

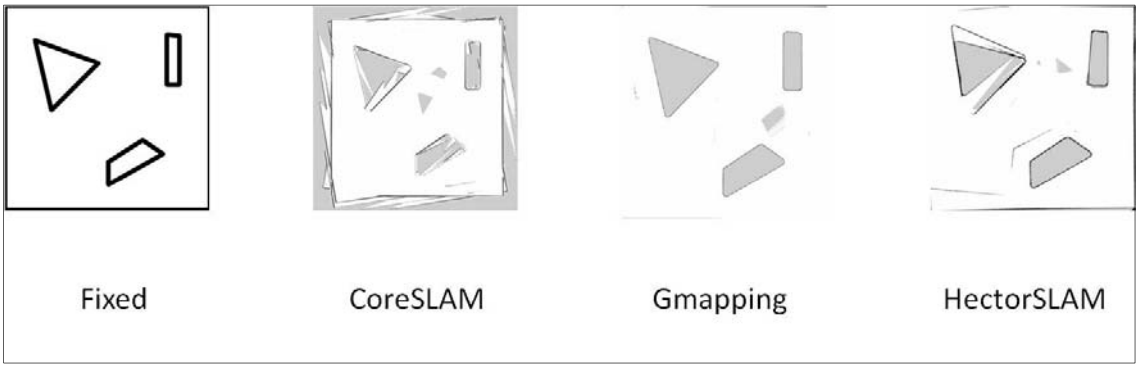

Figure 13. Map Three and Generated Map of Each Algorithm.

4.4. Image Registration of Ground Truth and Generated Maps

MATLAB (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language with many functions and libraries. This research uses the image registration tool to compare the generated maps to the ground truth map.

Image registration, the process of aligning two images of the same scene, is then used to align the ground truth map image and the generated map image for a multimodal comparison of the two images. An intensity-based automatic image registration process requires a pair of images, a metric, an optimizer, and a transformation type in order to align one image with another. The pair of images is the ground truth map image and the generated map image.

The metric defines the image similarity metric for evaluating the accuracy of the registration. The optimizer defines the methodology for minimizing or maximizing the similarity metric.

The transformation type defines the type of 2-D transformation that brings the misaligned image (called the moving image or the generated image) into alignment with the reference image (called the fixed image or the ground truth image). Four transform types exist: affine, rigid, similar, and translation.

The image registration process begins with the transform type you specify and an internally determined transformation matrix. Together, they determine the specific image transformation that is applied to the moving image with bilinear interpolation.

Next, the metric compares the transformed moving image to the fixed image and a metric value is computed.

Finally, the optimizer checks for a stop condition. A stop condition is anything that warrants the termination of the process. In most cases, the process has reached a point of diminishing returns or it has reached the specified maximum number of iterations. If there is no stop condition, the optimizer adjusts the transformation matrix to begin the next iteration.

The following sections discuss the resulting aligned images produced with the four transformations, i.e., affine, rigid, similar, and translation, and the three algorithms. The maps will contain three colors: magenta, green, and black. The magenta represents the intensity of the ground truth image, the green represents the intensity of the SLAM generated map, and the black represents where both images align or are the same.

4.4.1. CoreSLAM

Figures 14, 15, and 16 were produced using two images, (1) the ground truth map images, and (2) the Stage simulated generated map images from the CoreSLAM algorithm, as inputs to MATLAB's image registration function. In general, image registration overlays or aligns the generated map on to the ground truth map image to compare the two images. The image registration tool is executed with the four different transform types: affine, translation, rigid, and similar. Figures 14, 15, and 16 display the output images of the four transformations with the MATLAB code.

The generated maps show up with more intensity than the ground truth maps due to CoreSLAM producing multiple edges (green shaded areas) along the exterior portion of the map. While the ground truth and generated maps are similar in nature, there are few overlapping points because there is very little black, which shows points where the two images are identical.

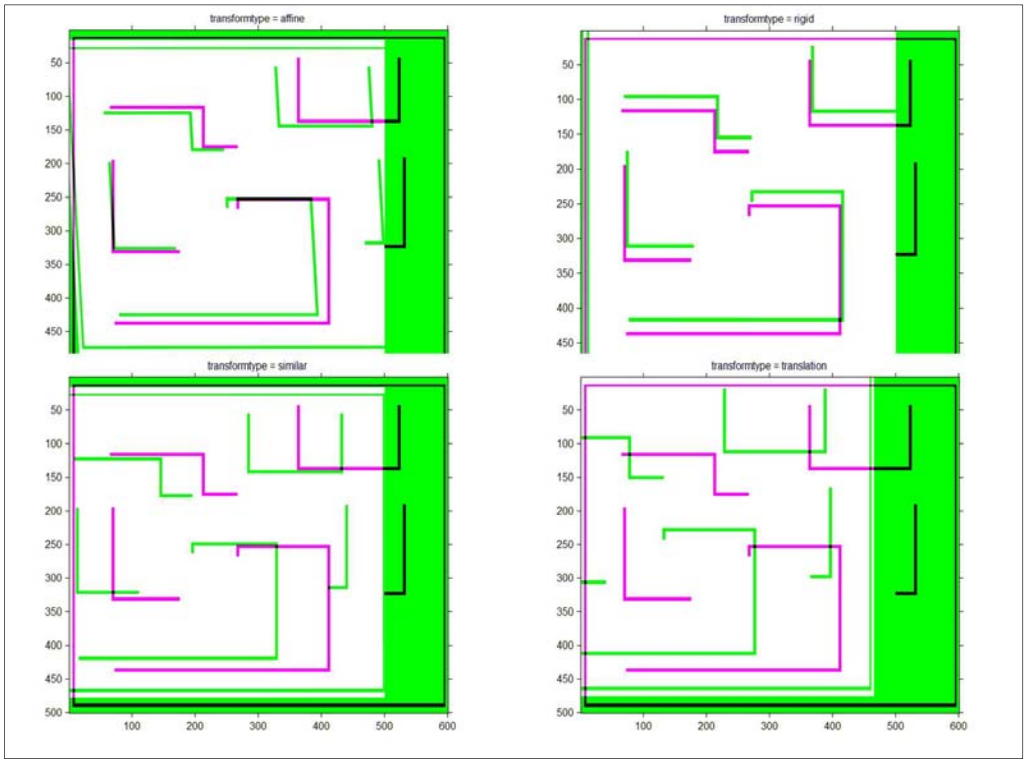

Figure 14. Image Registration of Map One and CoreSLAM Generated Map.

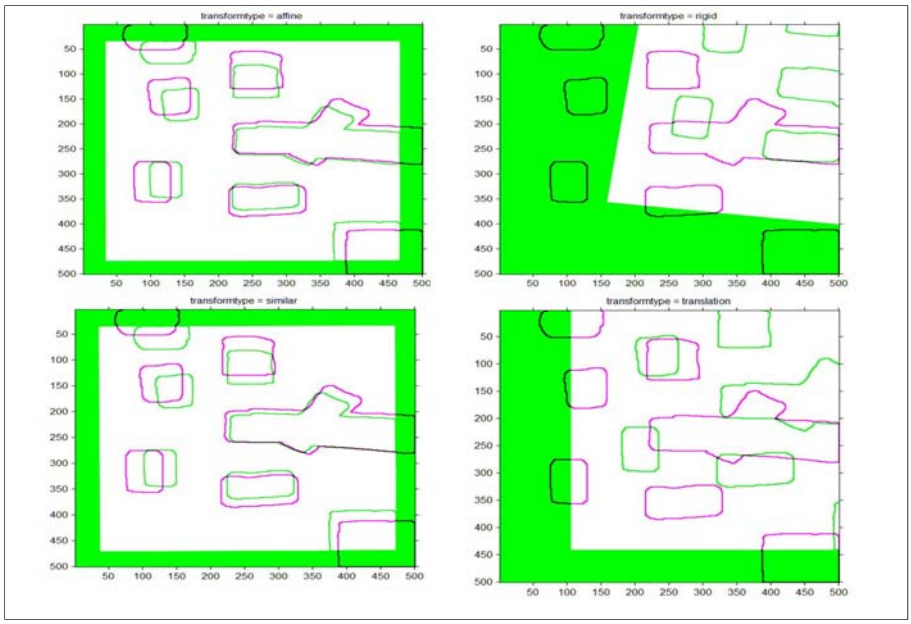

Figure 15. Image Registration of Map Two and CoreSLAM Generated Map.

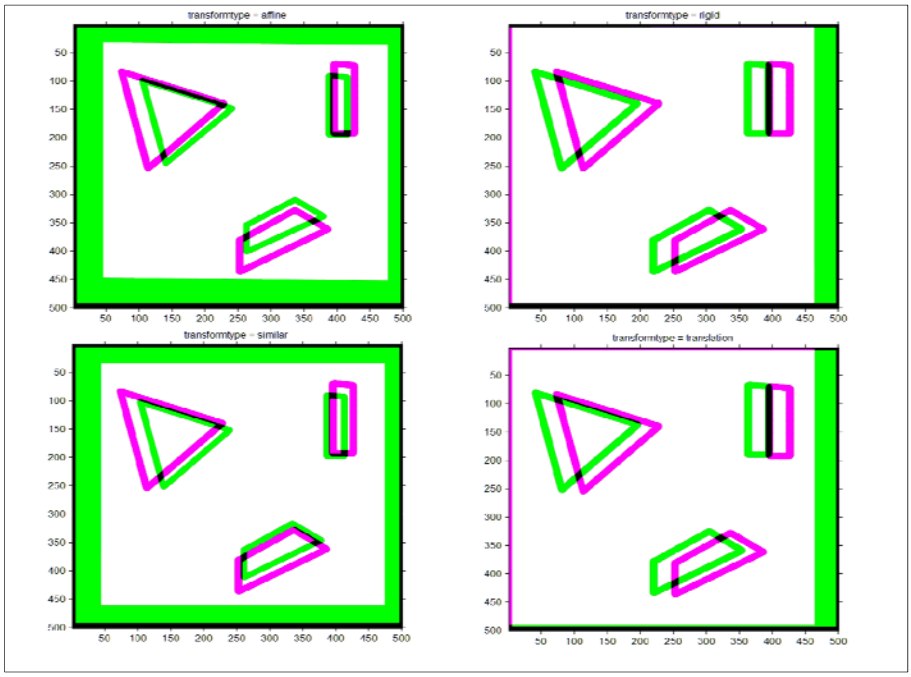

Figure 16. Image Registration of Map Three and CoreSLAM Generated Map.

4.4.2. Gmapping

Figures 17, 18, and 19 were produced using two images; the ground truth map images and the Stage simulated generated map images from the Gmapping algorithm in MATLAB's

image registration function. The magenta in the image of Figure 17 represents the intensity of the ground truth image, the green represents the intensity of the SLAM generated map, and the black represents where the images overlap. In the upper left of Figure 18, all transform types align all most perfectly. Transform type affine has a small amount of green in the bottom indicating that the ground truth map of Map Two has a slightly higher intensity than the generated map. Transform type translation (bottom right) has a small amount of green on the left and the bottom of the map, indicating the ground truth map of Map Two has a slightly higher intensity than the generated map. In Figure 19, the magenta is stronger in all transformations, indicating that the generated map has a higher intensity than the ground truth of Map Three. The black shows where the images align.

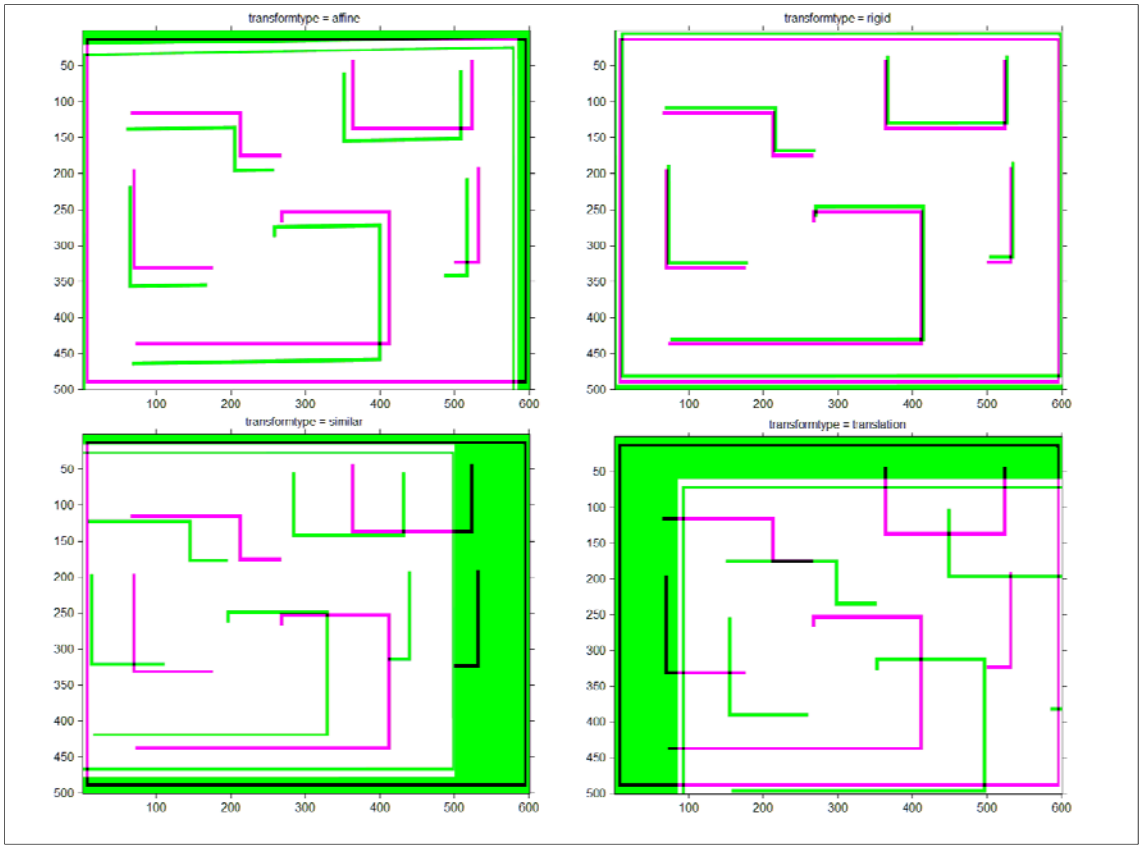

Figure 17. Image Registration of Map One and Gmapping Generated Map.

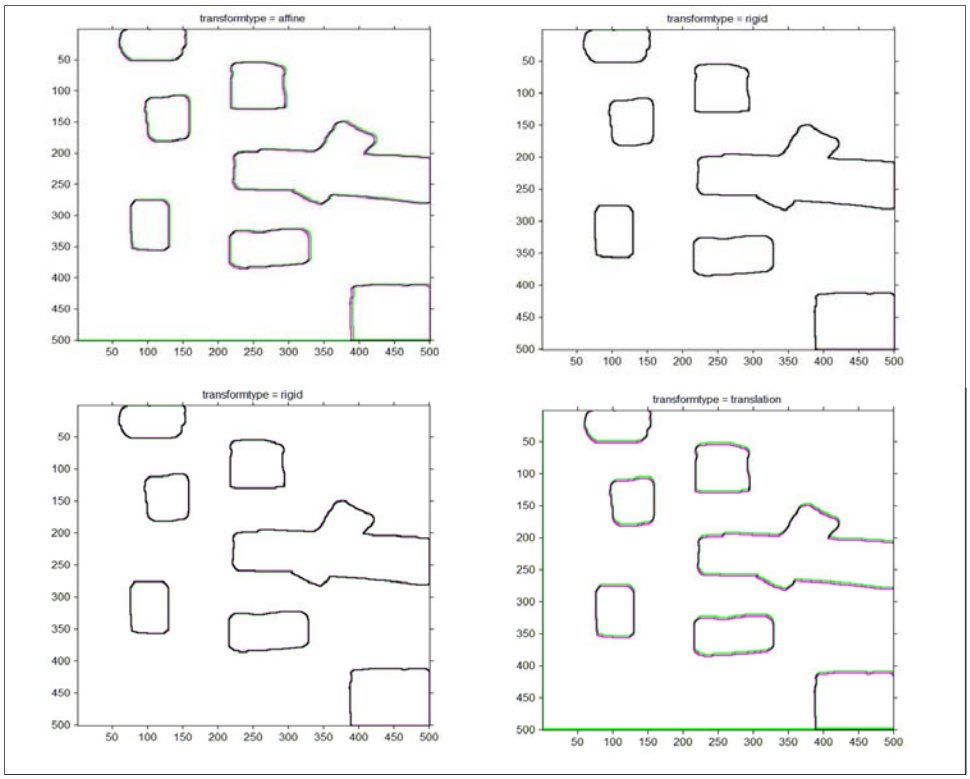

Figure 18. Image Registration of Map Two and Gmapping Generated Map.

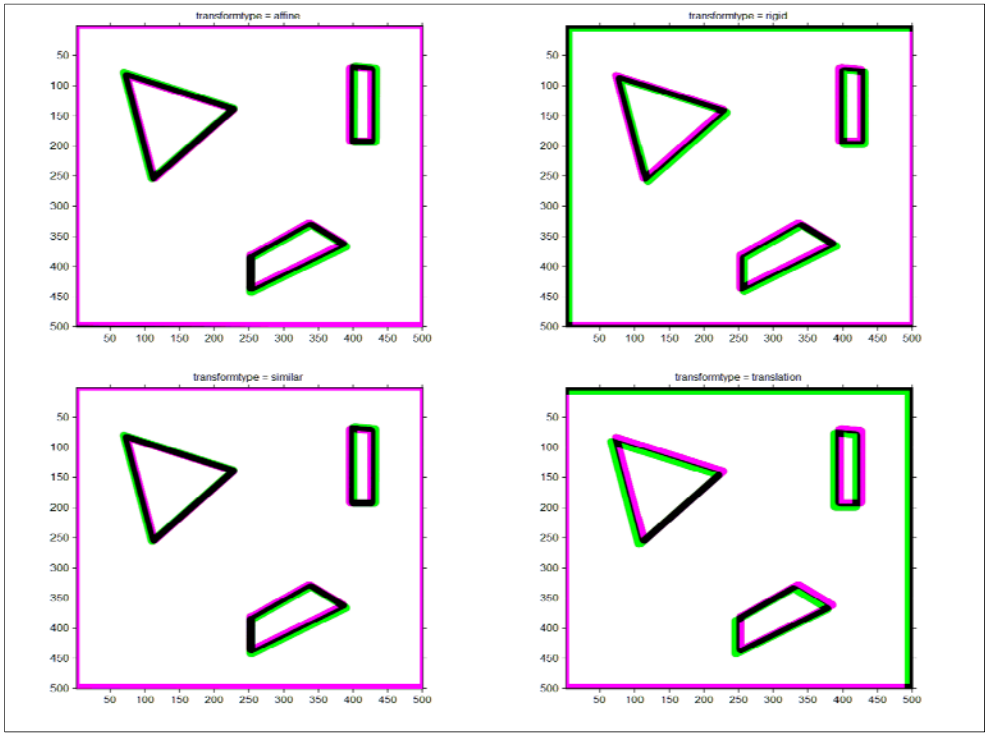

Figure 19. Image Registration of Map Three and Gmapping Generated Map.

4.4.3. Hector SLAM

Figures 20, 21, and 22 were produced by using the ground truth maps and the Stage simulated generated maps with the Hector SLAM algorithm as input in MATLAB's image registration function. The image registration tool was again processed with the four different transform types: affine, translation, rigid, and similar. Figure 20 shows the generated map has more intensity than the ground truth map due to the alignment being off and the degree of difference in the two maps. While the maps are similar in nature, they have few overlapping points. Figures 21 and 22 show black more than magenta and green indicating that the two images have very little differences. Figure 20 has more green with all transform types. Figure 21 shows green around the exterior due to the processing of image. In Figures 21 and 22, the rigid (upper right) and translation (lower right) transforms produce near perfect alignments.

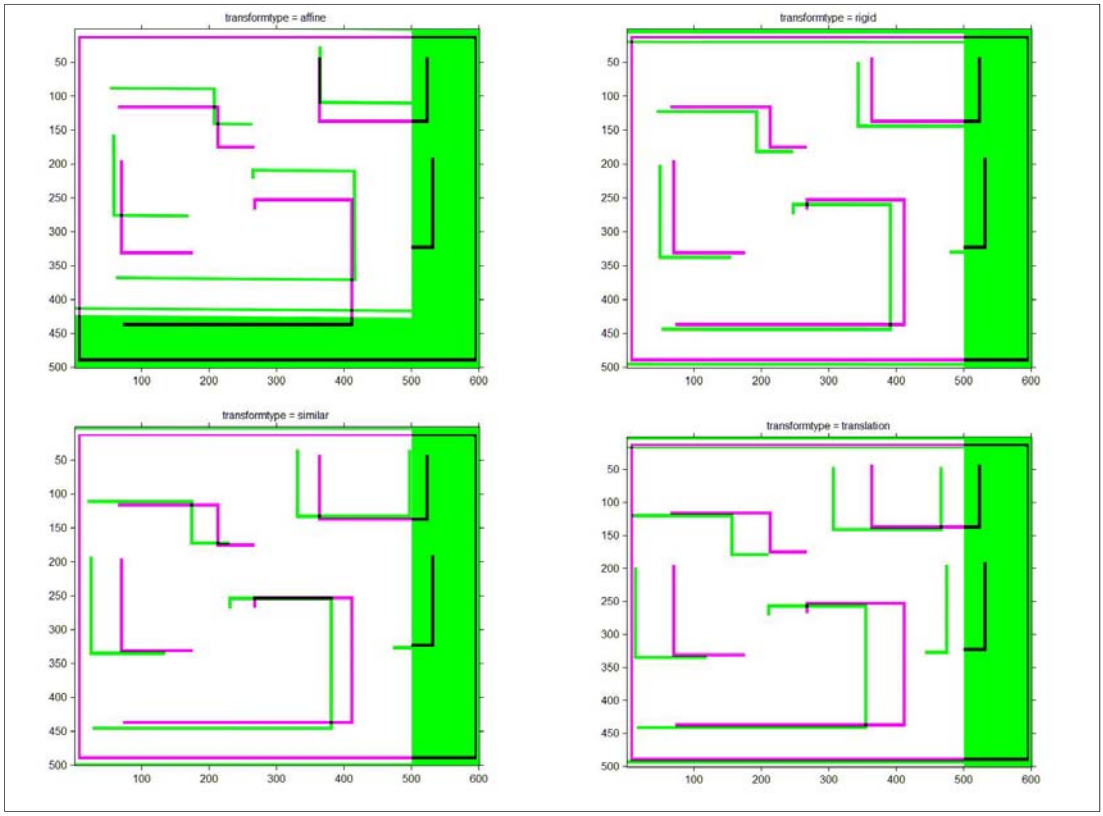

Figure 20. Image Registration of Map One and Hector SLAM Generated Map.

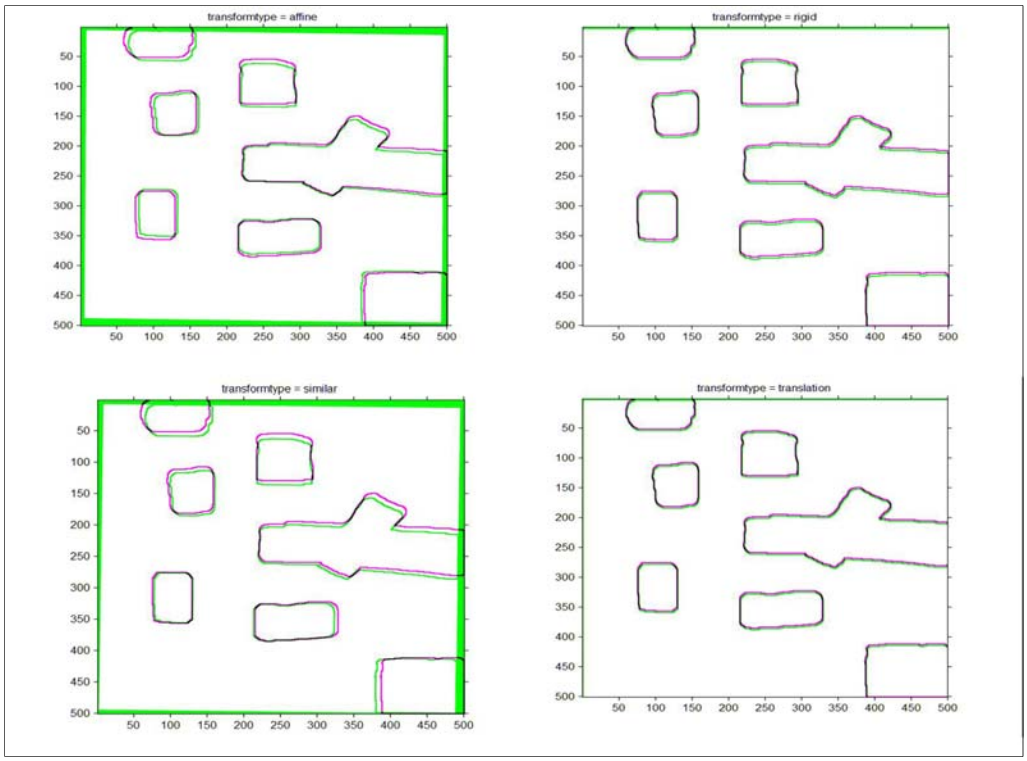

Figure 21. Image Registration Map Two and Hector SLAM Generated Map.

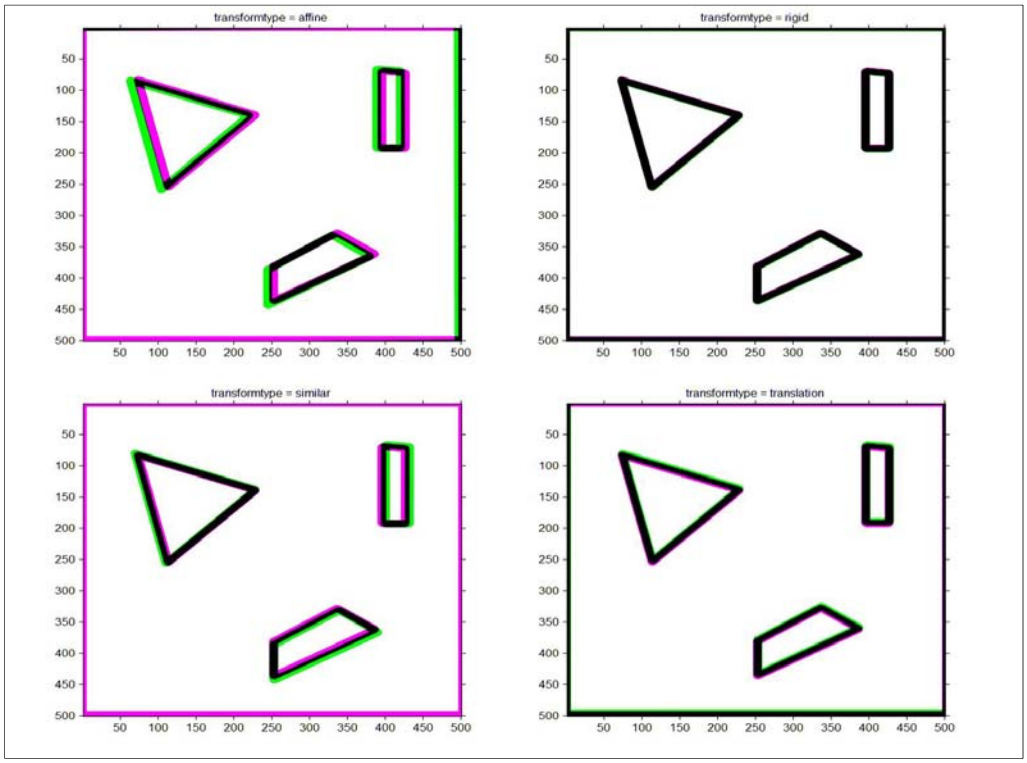

Figure 22. Image Registration of Map Three and Hector SLAM Generated Map.

4.4.4. Comparisons of Ground Truth Maps and Stage 4.1.1 Generated Maps

For a quantitative measure between the ground truth maps and the generated simulator maps, the Hausdorff Distance is calculated. The Hausdorff distance, by definition, is as follows: Given two finite sets A = (a_1….a_p) and B = (b_1…b_p), the distance is calculated as

$$H(A,B) = \max(h(A,B), h(B,A))$$

where

$$h(A,B) = \sup_{a \in A} \inf_{b \in B} \|a - b\|$$

‖ ‖ represents some underlying norm defined in the space of the two point sets, which is generally required to be an L_p norm, usually the L_2 or Euclidean norm. The function h(A, B) is called the directed Hausdorff distance from A to B. If A and B are compact sets, then

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

The Hausdorff Distance is calculated with the function h(A,B), which returns the distance of matrix A from matrix B. It identifies the point an element of A that is the farthest from any point in B and measures the distance from A to its nearest neighbor in B (comparing images using Hausdorff Distance paper).

Table 1 shows the HectorSLAM algorithm has the lower values of 15.5885, in the Hausdorff Distance Column. CoreSLAM has the second lowest value of 16.55. Gmapping has the highest value if 22.7156.

Table 2 shows the CoreSLAM algorithm has the lower value of 15.5563, in the Hausdorff Distance Column. HectorSLAM and Gmapping have an equal value of 16.8523.

Table 3 shows the HectorSLAM algorithm has the lower values of 14.1067, in the Hausdorff Distance Column. CoreSLAM having the second lowest value of 18.13857. Gmapping has the highest value of 19.2354.

Based on all the results, HectorSLAM outperforms both CoreSLAM and Gmapping for Map One and Map Three. CoreSLAM outperforms HectorSLAM and Gmapping for Map Two while HectorSLAM and Gmapping are tied for Map Two.

Table 1. Hausdorff Distance for Map One.

	Map Size	Hausdorff Distance
CoreSLAM	600 × 500	16.55
Gmapping	600 × 500	22.7156
Hector SLAM	600 × 500	**15.5885**

Table 2. Hausdorff Distance for Map Two.

	Map Size	Hausdorff Distance
CoreSLAM	500 × 500	**15.5563**
Gmapping	500 × 500	16.8523
Hector SLAM	500 × 500	16.8523

Table 3. Hausdorff Distance for Map Three.

Hector SLAM	Map Size	Hausdorff Distance
CoreSLAM	500 × 500	18.13847
Gmapping	500 × 500	19.2354
Hector SLAM	500 × 500	**14.1067**

CHAPTER 5

PHYSICAL TESTING EVALUATION

This chapter provides the evaluation of three laser-based SLAM algorithms, HectorSLAM, Gmapping, and CoreSLAM, implemented on the mobile robot, the Coroware Explorer. The tele-operated robot explored two physical test areas (ground truth maps) and generated a map of each area with each algorithm. Comparisons of the ground truth maps and the generated maps are discussed below.

5.1. Evaluated Algorithms

The algorithms used for physical testing are the same as those discussed in Section 4.1. The ultimate goal is to have the Explorer navigate each of the test areas and build a map of the unknown environment with each of the three algorithms. Each algorithm takes scan data from the Laser Range Finder attached to the Explorer and outputs a generated map.

5.2. Test Areas

Two areas were selected for ground truth maps. Figure 23 is the ground truth map of the foyer, break room, and mail room of Building 3296 at the ERDC, and Figure 24 is the ground truth map of the basement area of Building 3296 used to perform the physical test. The tele-operated Explorer navigated each area, and a map was generated simultaneously using the input from the Laser Range Finder and the three algorithms. The robot was networked with a laptop running UBUNTU and ROS Fuerte with each of the three algorithms. Each area was traversed with one of the selected algorithms to produce the three SLAM maps. The rooms in the

basement were empty while the foyer, break room, and mail room had tables, chairs, file cabinets, a sink based, soft drink machines, snack machines, garbage cans, ice machine, and recyle bins. The foyer's entry (Figure 22) is made of glass and has a larger area with two glass doors and an entry leading to the hallway with entry to the breakroom and the mailroom. The basement (Figure 23) consists of a hallway with four rooms and another hallway. Rooms 1, 3, and 4 have one door, while Room 2 has two doors. All four rooms were empty. The hallway was about five-feet wide and 50 feet in length.

5.3. Generated Maps from Physical Testing

This section shows the generated maps produced while navigating the two test areas with the mobile robot. CoreSLAM and Gmapping performed poorly while HectorSLAM generated a recognizable map with many identifiable features.

5.3.1. CoreSLAM

Figure 25 depicts the generated map from the CoreSLAM for the basement, and Figure 26 depicts the foyer. CoreSLAM performed poorly on the physical robot and the produced maps were of poor quality and unrecognizable compared to the ground truth maps. This could be due to the fact that the four rooms in the basement were small and the robot had to make an almost 360-degree turn in the rooms. However, the map for the foyer was also unrecognizable, therefore, this theory may be incorrect.

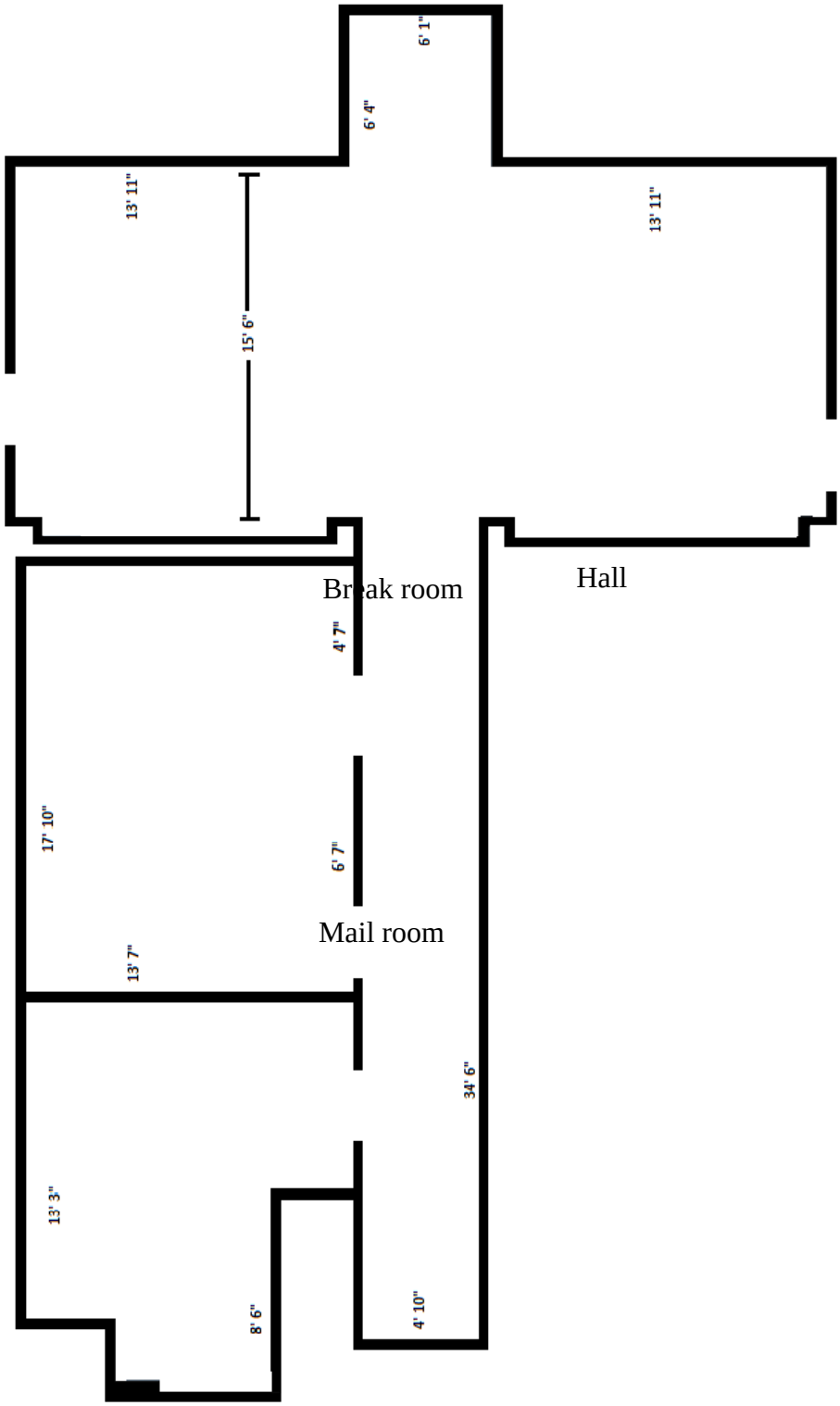

Figure 23. Ground truth map of foyer and Break Room.

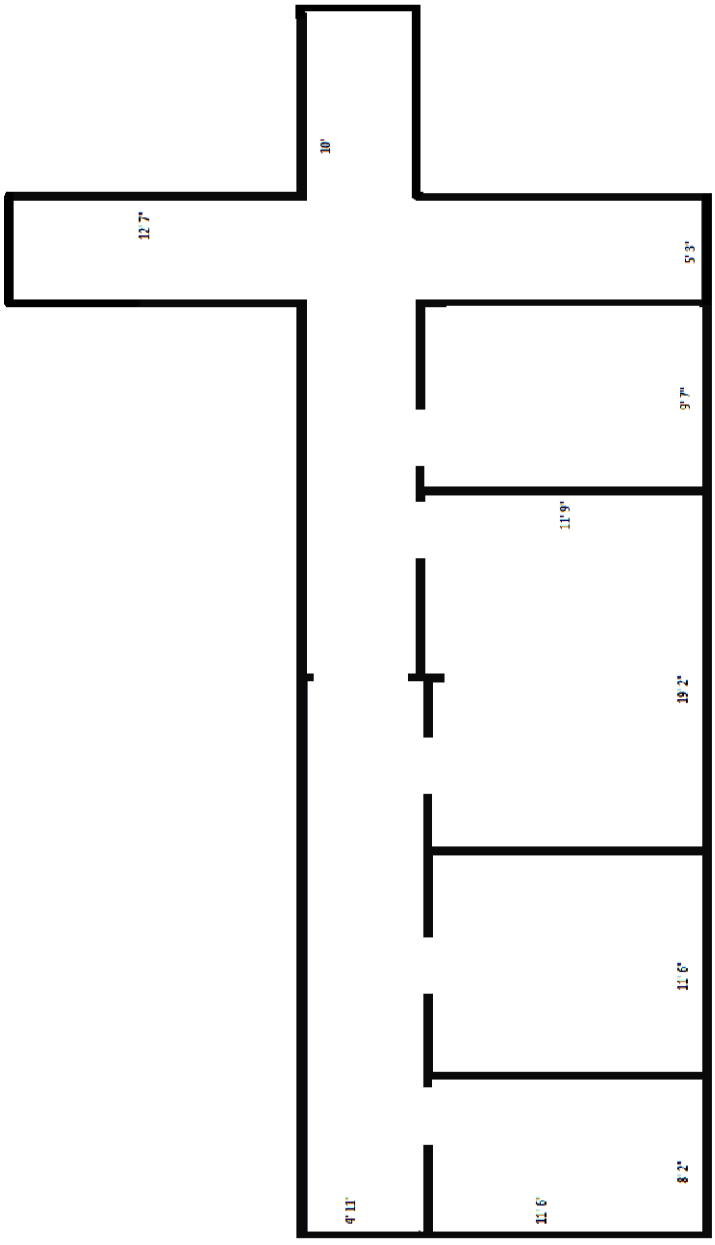

Figure 24. Ground truth map of basement.

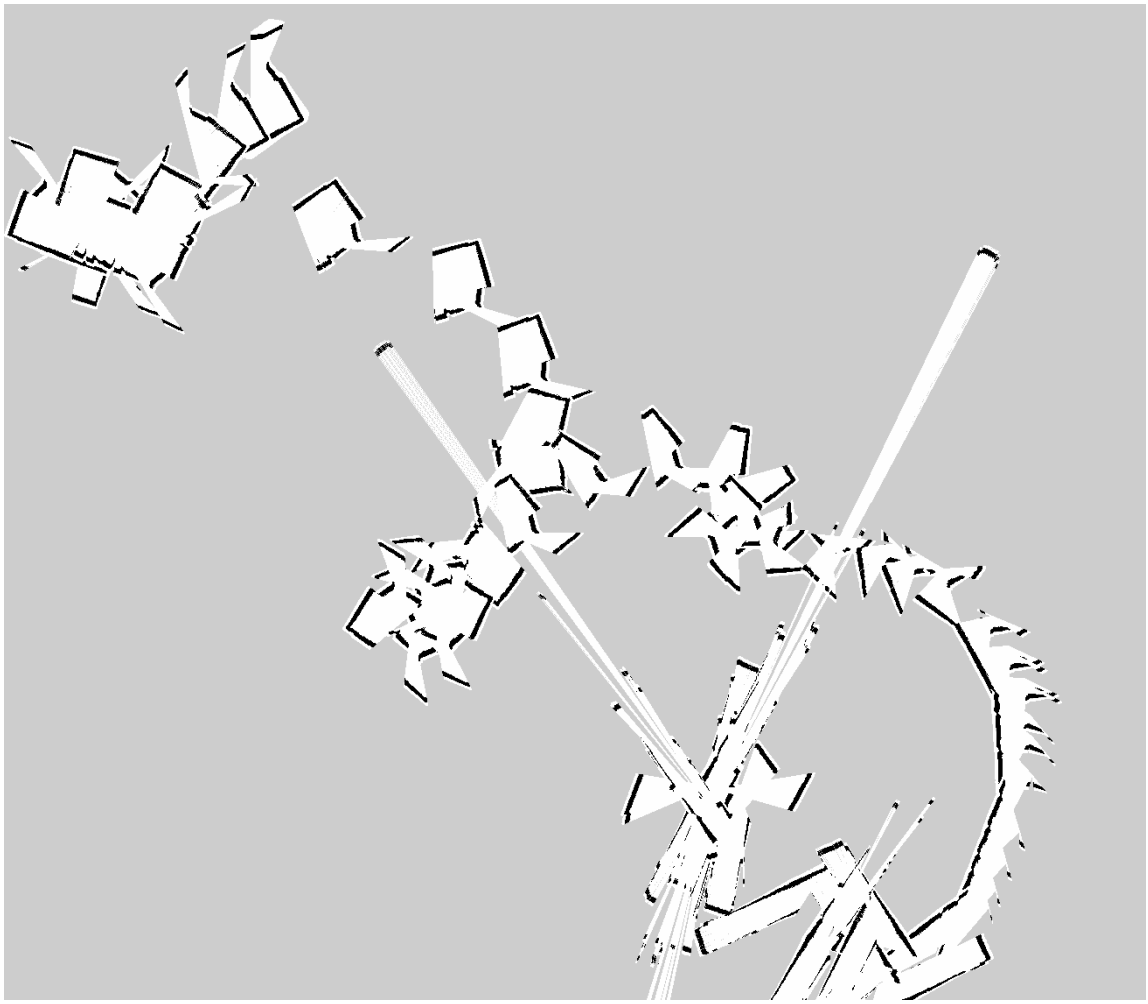

Figure 25. Generated CoreSLAM Map of Basement.

5.3.2. Gmapping

Figure 27 depicts the generated map from the Gmapping for the basement and Figure 28 the generated map of the foyer. Gmapping performed poorly on the physical robot, and the produced maps were of poor quality and unrecognizable compared to the ground truth maps.

Figure 26. Generated CoreSLAM Map of Foyer.

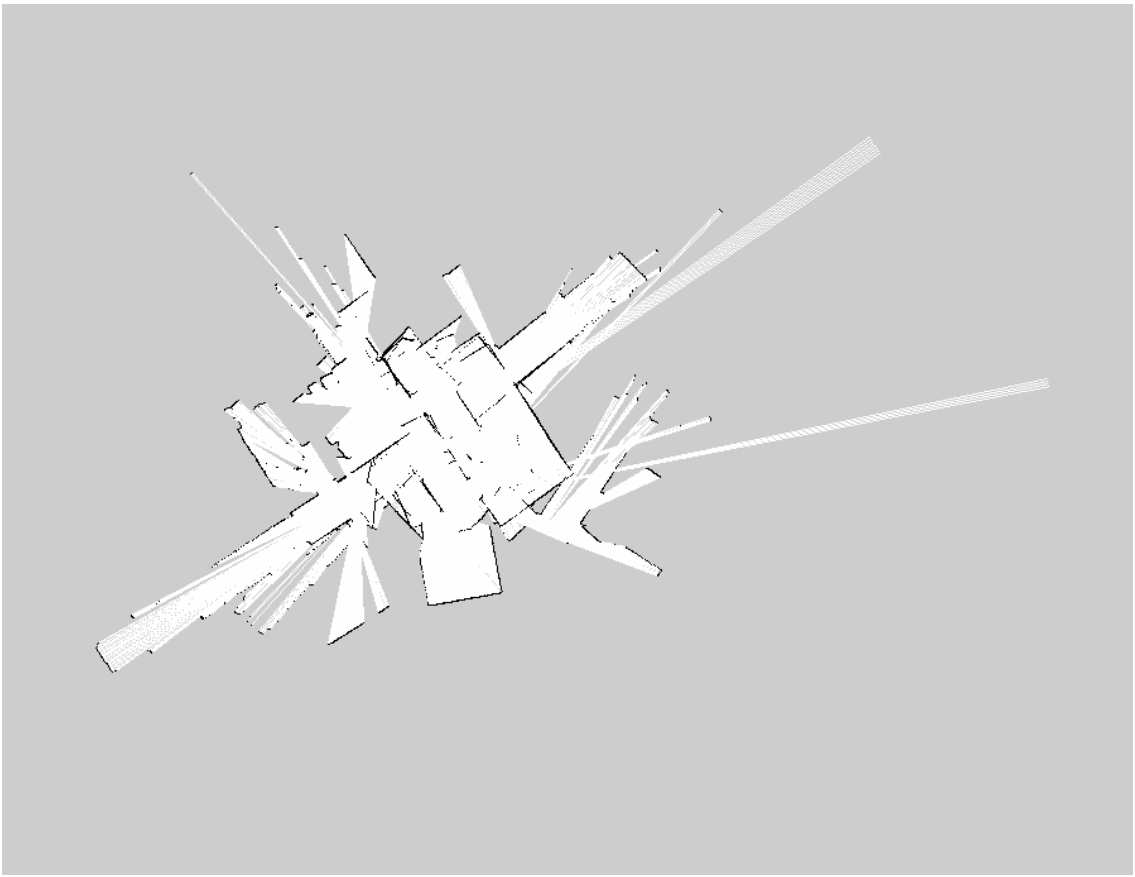

Figure 27. Generated Gmapping Map of Basement.

5.3.3. HectorSLAM

Figure 29 depicts the generated map from Hector SLAM for the basement, and Figure 30 is the generated map of the foyer. HectorSLAM performed efficiently on the physical robot, and the produced maps were good quality and recognizable, when compared to the ground truth maps. The generated map of the basement was somewhat distorted, but each of the four rooms and the hallway were identifiable.

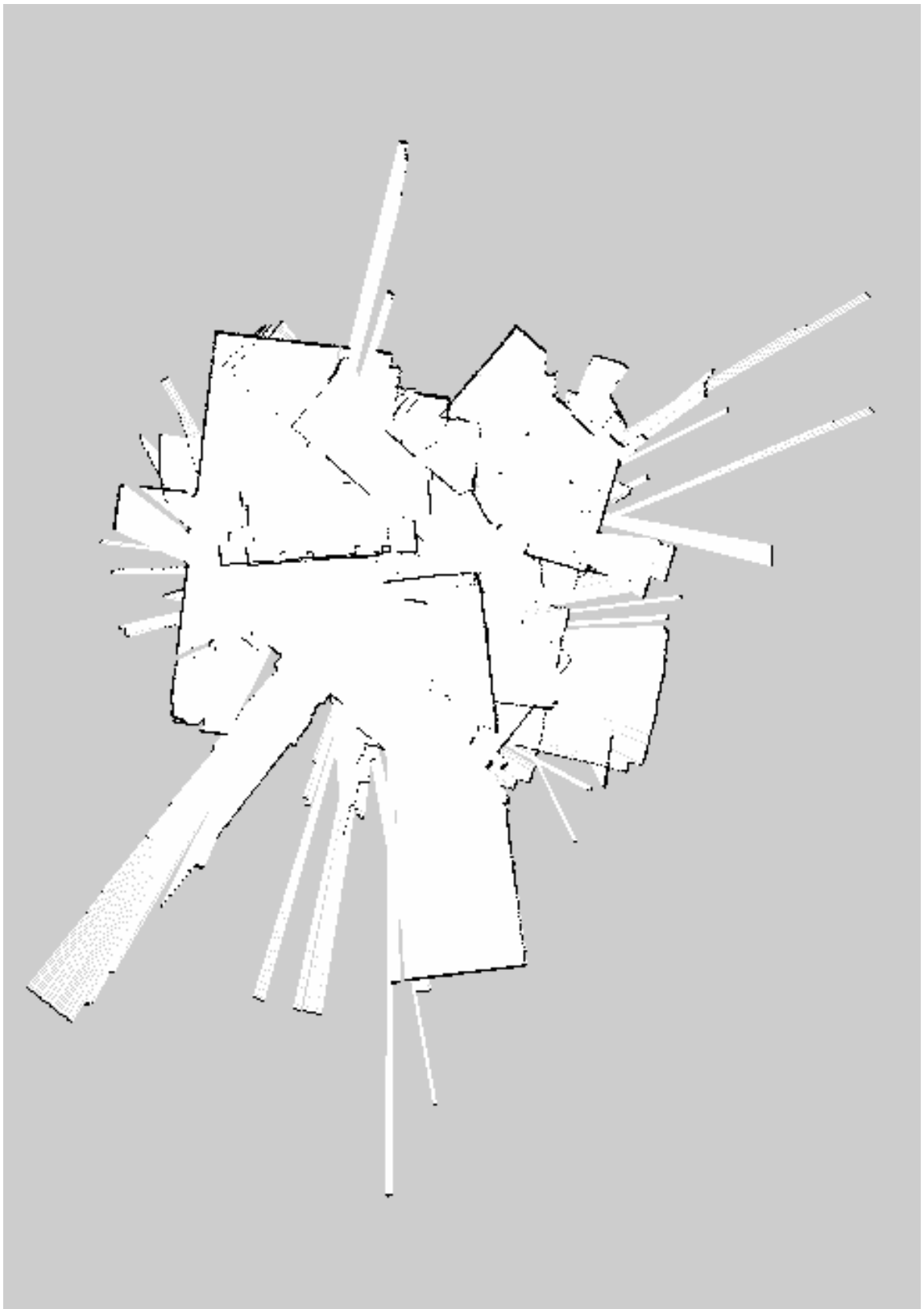

Figure 28. Generated Gmapping SLAM Map of Foyer.

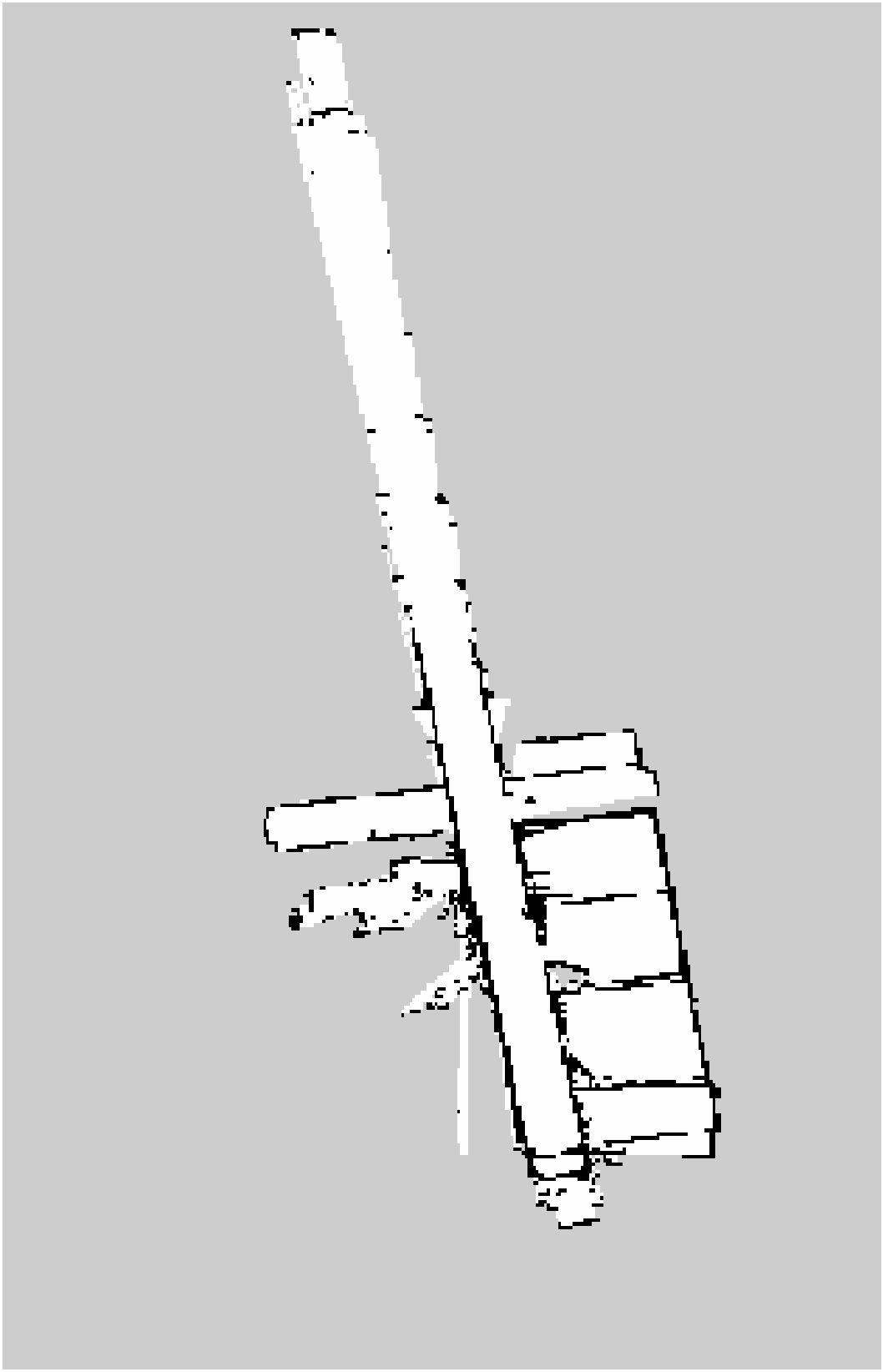

Figure 29. Generated Hector SLAM Map of Basement.

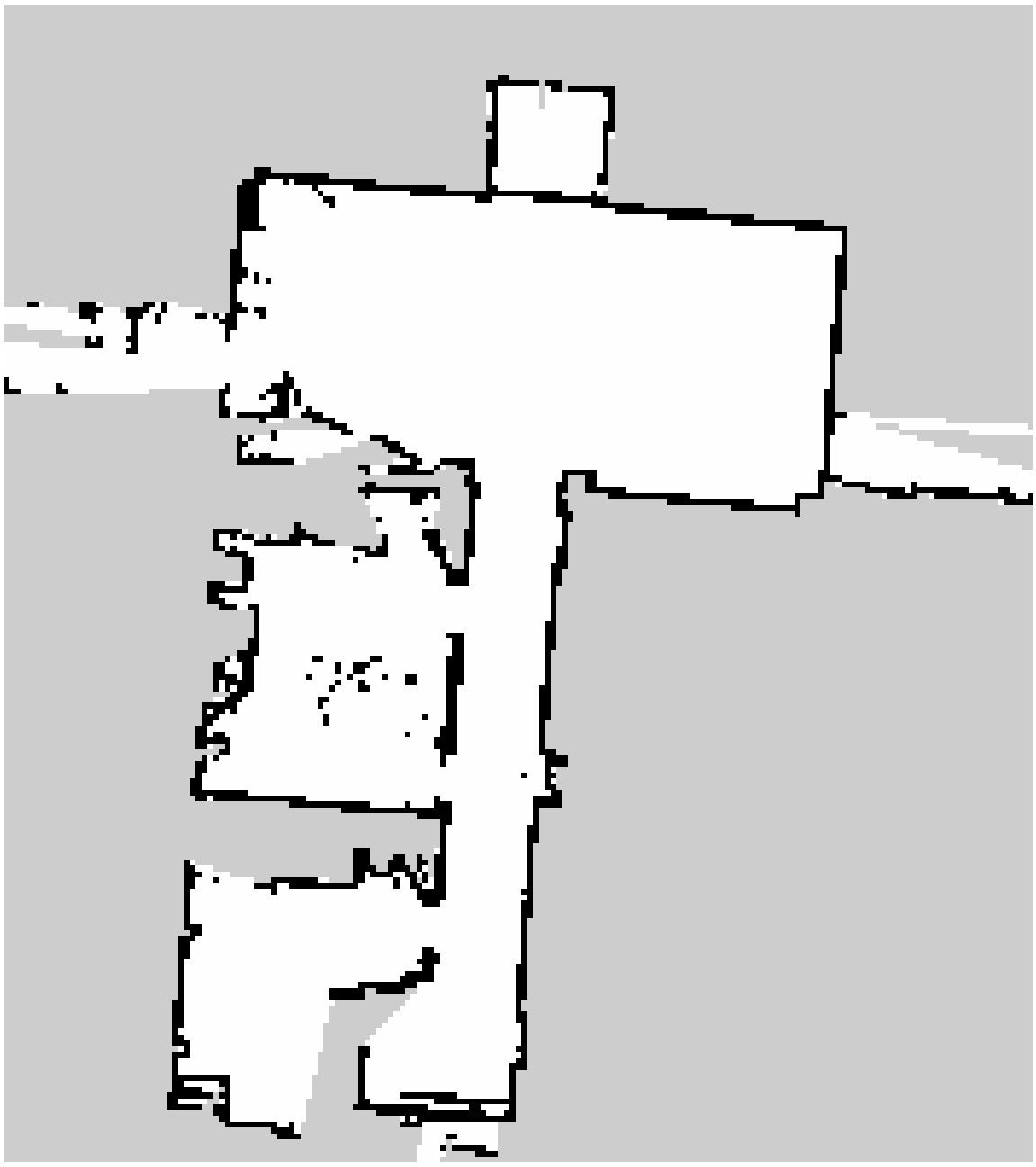

Figure 30. Generated Hector SLAM Map of Foyer.

5.4. Comparison of Ground Truth Map and Generated Map

Hector SLAM produced a better map than CoreSLAM and gMapping. Figure 31 shows the generated Hector SLAM map and actual photographs of the test site. Doors, windows, tables, chairs, and file cabinets can be identified in the generated map. Most of the major corners of the Hector SLAM map are recognizable. The gMapping generated map of the foyer had some recognizable features, but CoreSLAM had no recognizable features.

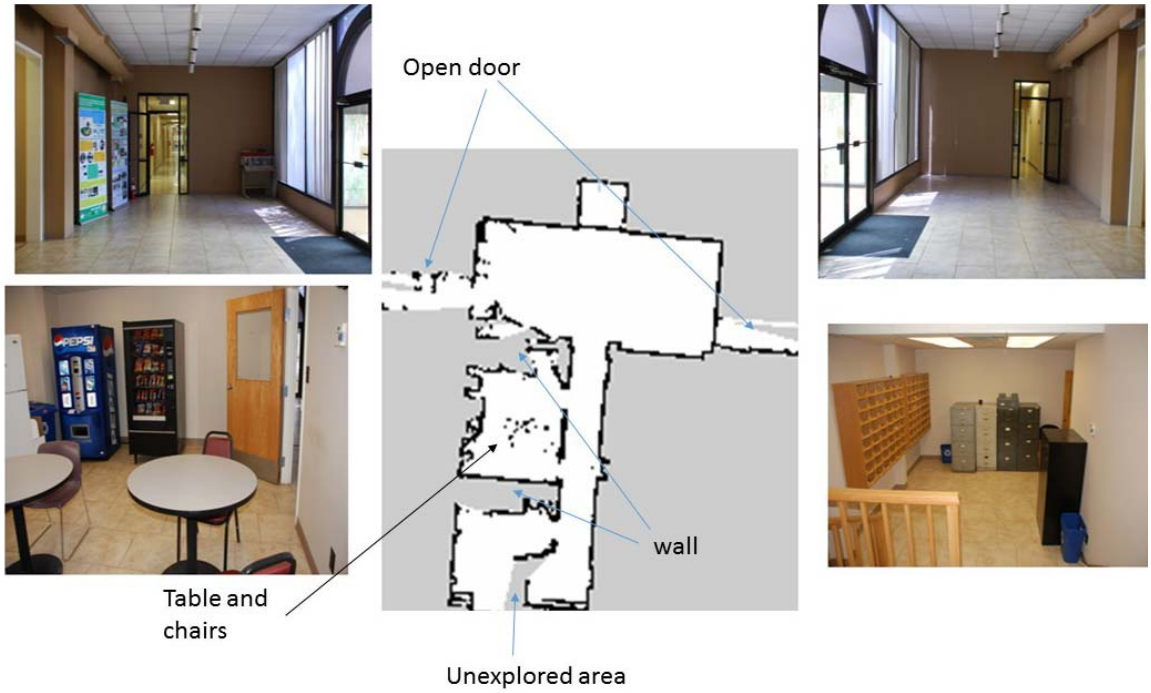

Figure 31. Generated Hector Slam Map and Pictures of actual foyer.

The map in the center is the generated map and the color images are actual photos of the foyer, breakroom, and mailroom. The generated map in the center has labels pointing to open door, tables and chairs, walls, and an unexplored area. The open door label points to an area where the Explorer did not travel, but the laser scans down the hallway off the foyer. Had the door been closed it would have scanned as a wall. The table and chairs label shows the dots on the floor where the table and chair legs are. The wall label is an area where there are actually

walls, cabinets, or door where the laser scanner cannot scan. The unexplored area, the grey area, is an area in the mail room where the laser scan did not scan.

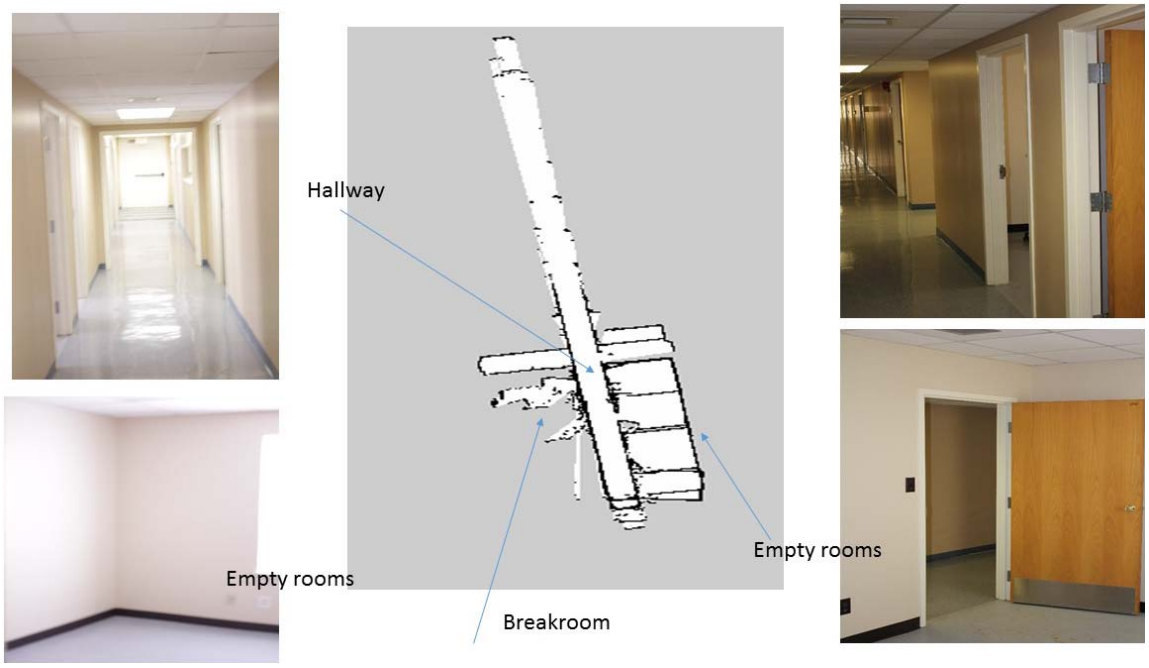

Figure 32. Generated Hector Slam Map and Pictures of actual basement.

The map in the center of Figure 33 is the generated map of the basement and the color images are actual photos of the basement, consisting of a hallway and four small rooms. The generated map in the center has labels pointing to empty rooms, the hallway and a breakroom not traversed by the robot but scanned by the laser scanner. As depicted in Figure 32, the rooms are small and the hallway narrow, and the lighting varies from light to dark. Too much light could affect the scan, so paper and blinds were placed over the window to control the amount of light. I believe the smallness of rooms and the narrowness of the hallway had an effect on the quality of the map.

5.5. Results

This chapter provided physical testing of the Explorer in the foyer area and the basement. The Explorer performed better in the foyer, and a higher quality map was generated. This Hector SLAM and Coroware Explorer research increases the ability of the ERDC to explore and map unknown areas and aids in increasing the level of autonomy of current unmanned vehicles used during the exploration of unknown areas. This implementation and testing provides further validation of the Hector Slam algorithm for real world applications.

CHAPTER 6

LOCALIZATION AND PATHFINDING

This chapter localizes the Explorer to a position in the generated grid-based map produced by Hector Slam in Chapter 5 using the Adaptive Monte Carlo Localization Package (AMCL) for a robot moving in 2-D. AMCL produces the positon of the moving robot in the map. Several maps are produced at resolutions with three different physical locations. The results of the robot's path-finding ability with A* algorithm package was collected for a path between two points, in both the generated SLAM map and the physical site.

6.1. A* Pathfinding Package

The goal of this package is for the Explorer to create a path based on a grid-based map produced by the mobile Explorer. Several maps are generated at varying resolutions of 100×100, 200×200, 300×300, 500×500, and 1000×1000 of three different test sites. The Explorer travels around the map from start node A, provided by the AMCL package, and to destination node B, provided by the user and stored and passed to the A* package. A path is produced by the A* algorithm, and then the path is timed and traversed.

6.2. Requirements

The mobile robot must be able to localize itself in the grid-based map produced by HectorSLAM and traverse a path produced by the A* algorithm. The package takes as input a grid-base map, a start location, and a destination location. The robot must travel the path produced by A*. The robot must also traverse the map, storing a start location, and travel to a

destination. The actual distance between the destination point B of the SLAM map and point B of the physical map was measured, recorded, and analyzed.

6.3. Analysis and Design

The analysis and design phase consists of the analysis and design to localize the Explorer to a grid-based map and produce a path in the grid-based map. The Corobot A* package allows the Explorer to traverse a path in a grid-base map autonomously. The A* Package has the following steps.

1. Take as input a grid-based map produced with HectorSLAM.
2. Convert the map to an array of 0s and 1s based on map resolution. 0 is free. 1 is occupied.
3. Tele-operate the mobile robot to localize to the map and get the current positon of the robot.
4. Store the current position as the start location.
5. Enter the destination position.
6. Calculate the path.
7. Convert the start and destination position to cells in the map.
8. Set start time for path traversal.
9. Mobile robot travels autonomously to destination.
10. Set end time for path traversal.
11. Calculate time to traverse path.
12. Measure the distance between the starting and ending points.
13. Repeat process on maps of varying resolution.
14. Collect results on pathfinding at each resolution.

6.3.1 AMCL Overview

AMCL is a ROS package designed to provide a mobile robot its pose while traversing a known map. AMCL is a probabilistic localization system for a robot moving in 2-D (AMCL site). AMCL implements the adaptive (or KLD-sampling) Monte Carlo approach as described by Dieter Fox (AMCL site). AMCL takes as input a laser-based map and publishes the mobile robot's estimated pose in the map. AMCL subscribes to topics in Table 4.

Table 4. Subscribed Topic.

scan	sensor_msgs/LaserScan
tf	tf/tf_messages
initialpose	geometry_msgs/PosWithCovarianceStamped
Map	Navigation_msgs

AMCL publishes the amcl_pose, which is the pose of the mobile robot against the known map.

6.3.2 Corobot_A* Package

The Corobot A* Package implements the A* algorithm with the Manhattan Cost function. The package uses the robot pose, an x and y position in the map, provided by AMCL, to use as the start location for the path, and the user provides a random destination. The path between the start and destination location is calculated and provided. The robot then traverses the given path.

Three cost functions were considered: (1) the Manhattan Distance, (2) Euclidean Distance, and (3) the Chebyshev Distance.

The Manhattan Distance was implemented as the map is an NxN matrix, and four of the surrounding cells of the map were considered. The function is F = G + H, where G is the total distance to the current position and H represents the cost.

6.4. Implementation

While the robot is traversing the area, a map is created of the explored area. The robot is then localized to the map using AMCL, explores the area, and displays its pose on the map. The robot can be given a destination position to travel using its current position as the start location.

6.5. Testing

Three test sites were used to evaluate the ability of the Explorer to localize itself in a generated SLAM map and travel from a start location to a destination location. For each site, two cones were used to mark points A and B. The area around the cones were taped to marked points A and B. The area was traversed by the Explorer, and the SLAM map was generated. This map was used to localize the robot to the position A. The destination coordinate was entered into the A* package, and the difference between the two B points was measured and the time to traverse the path recorded.

6.5.1 Test Site One

Figure 33 depicts Test Site One, a 12-foot by 9-foot area that has walls on one side and two 6-foot tables as a wall for the 12-foot side. The 9-foot side has one 9-foot table and a board to enclose one side, and the other 9-foot side was a constructed wall. The x axis is marked and was determined to be the x-axis by driving the Explorer around, and the x position of the pose data increased, which is the northing in the map. The y axis is marked and was determined to be the y axis by driving the Explorer around, and the y position of the pose data increased.

Figure 34 depicts Test Site One without the cones and positions A and B marked with tape and the SLAM generated map with the two black squares representing points A and B in the SLAM generated map.

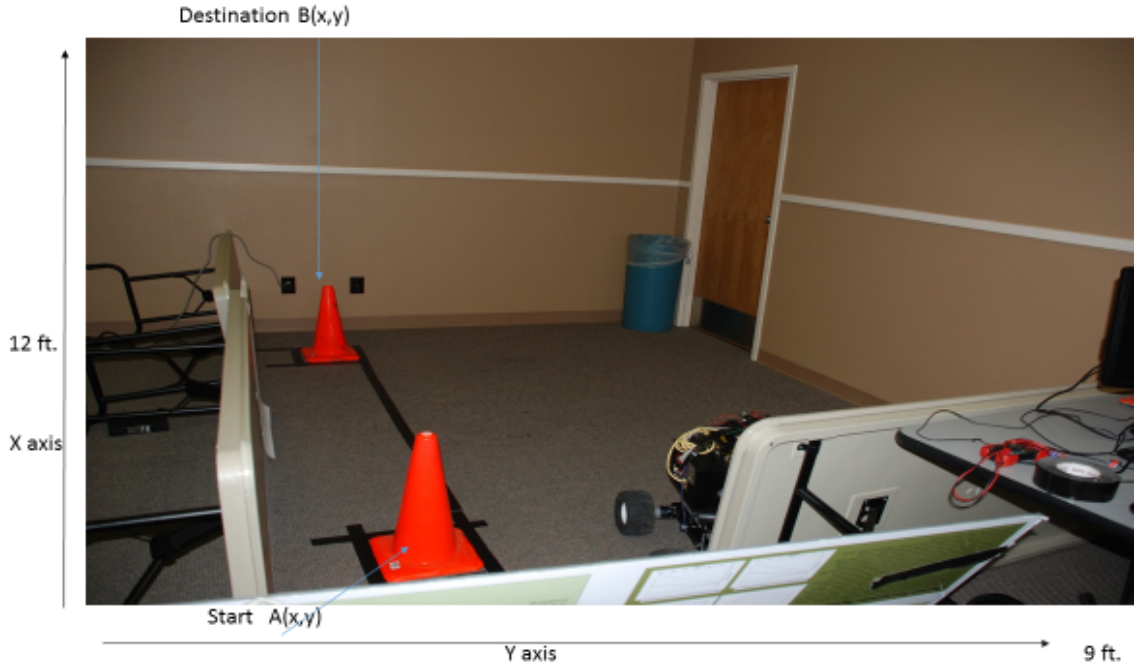

Figure 33. Test Area One with cones denoting points A and B.

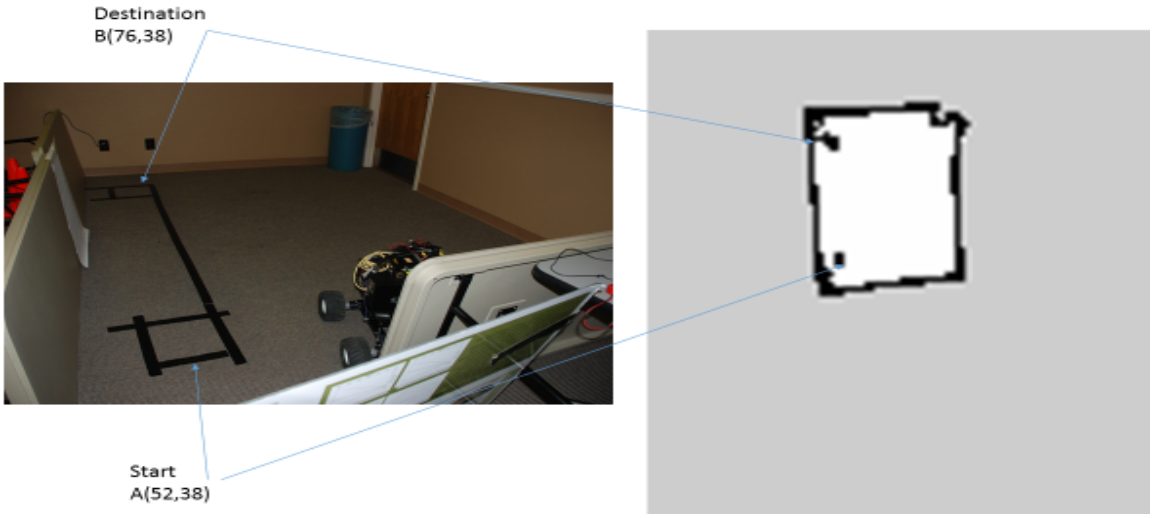

Figure 34. Test Site One without cones and the SLAM Generated Map.

Next, the generated map is imported into GIMP 2 and scaled to the resolution of the map to gather the x and y positions for each of the points marked in the previous steps. The two points shown in the Figure 35 are for a map of Resolution of 100×100. This step is repeated to gather the start and destination points for maps of resolution 200×200, 300×300, and 500×500.

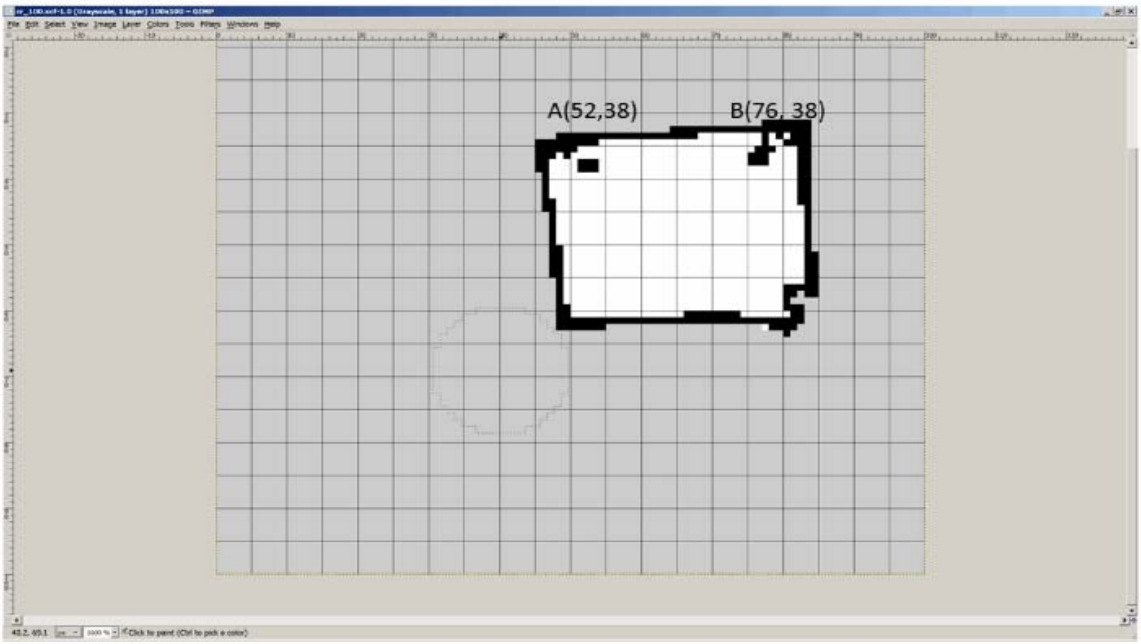

Figure 35. GIMP View with imported SLAM map.

Next, the Explorer and laptop are started with the correct setup to use the AMCL Package and the generated map of the test site without the cones. The robot is placed in the A position of the test site, then localized to the map, and the A* package is executed. The AMCL package provided a location of (40,60) and provided a B position for the target location. Table 5 shows the results of these runs for a map of Resolution 100×100. The distance from where the robot ended and the position of Point B in the physical map is shown in the Distance column, and the time to traverse the path is shown in the Time column.

When entering the target location, the A* package was executed until a B location close to the target destination B was found. Point (76, 38) was too large. Point (65, 40) and (65,50) are feasible and very close to the target destination point.

In the 100×100 resolution map, Table 5, the error was from 7 to 16 inches, with an average of 10.3 inches for the five runs. In the 200×200 resolution map, Table 6, the error was from 0 to 16 inches, with an average of 9.9 inches for the five runs. In the 300×300 resolution

map, Table 7, the error was from 0 to 18 inches, with an average of 9.2 inches for the five runs. In the 500×500, Table 8, resolution map, the error was from 2 to 18 inches, with an average of 10.2 inches for the five runs. Overall, for Test Site One, the 300×300 resolution performed best.

In Test Site One, when entering the generated map's point B, the robot could not find point B. The robot would try to travel up the wall and would have to be stopped. This led to finding a point B close to the Point B on the floor to enter. This was possibly caused by the area being too small and Point A and Point B being close together in the physical map. As we move to Test Site Two, a larger area was chosen.

Table 5. Results for 100×100 on Test Site One.

	Generated Map		AMCL			
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(52,38)	(76,38)	(60,40)	(65,40)	11.85	9.00
Run 2	(52,38)	(76,38)	(60,40)	(65,40)	15.51	12.00
Run 3	(52,38)	(76,38)	(60,40)	(65,40)	16.37	7.50
Run 4	(52,38)	(76,38)	(60,40)	(65,40)	13.50	16.00
Run 5	(52,38)	(76,38)	(60,50)	(65,50)	14.39	7.00

Table 6. Results 200×200 on Test Site One.

	Generated Map		AMCL			
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(154,141)	(177,148)	(90,110)	(90,110)	13.36	0
Run 2	(154,141)	(177,148)	(90,110)	(90,110)	14.20	0
Run 3	(154,141)	(177,148)	(90,110)	(90,110)	14.72	17
Run 4	(154,141)	(177,148)	(90,110)	(90,116)	16.35	16
Run 5	(154,141)	(177,148)	(90,110)	(90,116)	13.42	16

Table 7. Results for 300×300 on Test Site One.

	Generated Map		AMCL			
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(155,140)	(180,146)	(150,150)	(150,156)	20.57	16
Run 2	(155,140)	(180,146)	150,150)	(150,156)	15.01	18
Run 3	(155,140)	(180,146)	150,150)	(150,156)	14.46	4
Run 4	(155,140)	(180,146)	150,150)	(150,156)	13.52	8
Run 5	(155,140)	(180,146)	150,150)	(150,156)	15.70	0

Table 8. Results for 500×500 on Test Site One.

	Generated Map		AMCL			
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(250,243)	(250,241)	(250,250)	(250,256)	8.87	14
Run 2	(250,243)	(250,241)	(250,250)	(250,256)	9.83	2
Run 3	(250,243)	(250,241)	(240,250)	(250,257)	9.33	5
Run 4	(250,243)	(250,241)	(250,250)	(250,256)	8.99	12
Run 5	(250,243)	(250,241)	(250,250)	(250,256)	8.72	18

6.5.2 Test Site Two

Test Site Two is an area approximately 15 feet by 24 feet with bookshelves, several desks, a copy machine, a garbage can, and several small file cabinets, (Figure 37). The hallway is blocked by a piece of board to make a wall. The map was generated for Test Site Two with Hector SLAM with the perimeter of the two cones taped on the floor in order to mark points A(x, y) and B(x, y) in the actual test site.

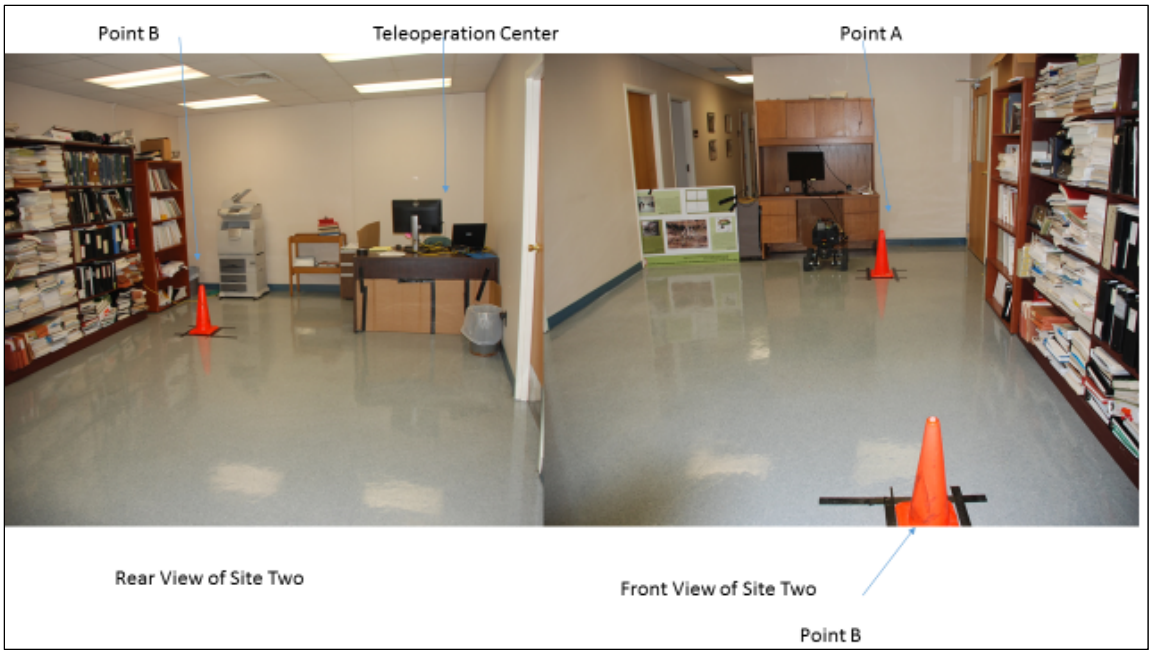

Figure 36. Front and Rear View of Test Site Two with cones.

Next, the generated maps are imported into GIMP 2 and scaled to resolution size. For Test Site Two, the resolutions tested were 200×200, 300×300, 500×500, and 1000×1000. Resolution 100×100 produced a poor quality map and was not on the canvas. In Figure 36, Point A is the start location, and Point B is the destination location.

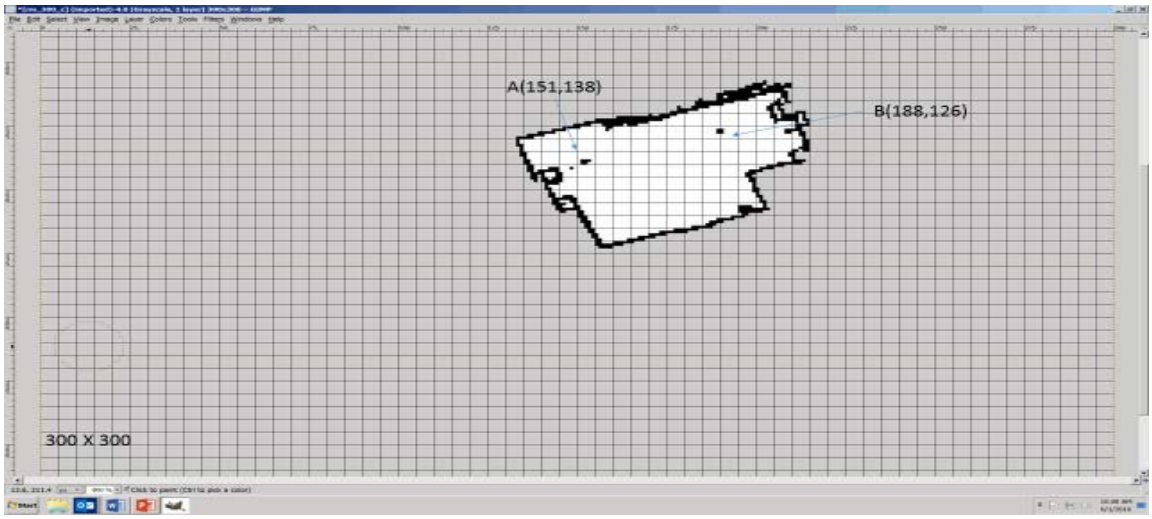

Figure 37. Test Site Two Resolution 300×300.

Next, the Explorer and laptop are started with the correct setup to use the AMCL Package and the generated map of the test site without the cones. The Explore was placed in the start position A of the test site, then localized to the map, and the A* package was executed. The AMCL package provided the starting point A(x,y) , and the user provided B(x,y) position for the destination location. No results were collected for Resolution 1000×1000. Each time the A* Package was executed, the package had a memory core dump.

Table 9. Results for 200×200 on Test Site Two.

	Generated Map		AMCL	Entered		
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(107,92)	(145,99)	(100,100)	(115,100)	21.67	0
Run 2	(107,92)	(145,99)	(100,100)	(115,100)	21.03	30
Run 3	(107,92)	(145,99)	(100,100)	(115,100)	20.54	24
Run 4	(107,92)	(145,99)	(100,100)	(115,100)	22.23	36
Run 5	(107,92)	(145,99)	(100,100)	(115,100)	23.48	48

Table 10.Results for 300×300 on Test Site Two.

	Generated Map		AMCL	Entered		
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(151,138)	(188,126)	(150,150)	(150,164)	19.70	1
Run 2	(151,138)	(188,126)	(140,150)	(140,164)	19.51	12
Run 3	(151,138)	(188,126)	(150,150)	(150,164)	19.16	1
Run 4	(151,138)	(188,126)	(150,150)	(150,164)	19.20	0
Run 5	(151,138)	(188,126)	(140,150)	(140,164)	19.56	0

Table 11. Results for 500×500 on Test Site Two.

	Generated Map		AMCL	Entered		
	A	B	A	B	Time (sec.)	Distance (in.)
Run 1	(256,241)	(294,241)	250,250)	(264,250)	21.04	2
Run 2	(256,241)	(294,241)	(250,250)	(264,240)	20.53	12
Run 3	(256,241)	(294,241)	(240,250)	(264,250)	19.34	0
Run 4	(256,241)	(294,241)	(240,250)	(264,250)	19.63	0
Run 5	(256,241)	(294,241)	(240,250)	(264,250)	19.71	0

In the 200×200 resolution map, Table 9, the error was from 0 to 48 inches, with an average of 27.6 inches for the five runs. In the 300×300 resolution map, Table 10, the error was from 0 to 12 inches, with an average of 2.8 inches for the five runs. In the 500×500 resolution map, Table 11, the error was from 0 to 12 inches, with an average of 2.8 inches for the five runs. Overall, for Test Site 2, the 300×300 and 500×500 resolution performed best.

In this set of tests, it is obvious that changing the resolution decreases some of the error. Creating a map of resolution 100×100 was unsuccessful. The map that was produced was of poor quality. As in Test Site One, the entered Point B was tweaked by driving the robot around to find a feasible Point B. The Point B of the generated map and the one collected were extremely different, but once Point B was found, the error was low.

6.5.3 Test Site Three

The map was generated for Test Site Three, an area approximately 15 feet by 24 feet with Hector SLAM with the perimeter of the two cones taped on the floor in order to mark Points A(x, y) and B(x, y) in the actual test site. Test Site Three is the same room as Test Site Two, but the room was rearranged. The copy machine was moved, and the desk was moved from one wall to another. Figure 38 depicts Test Site Three.

Figure 38. Test Site Three with cone.

Next, the generated maps were imported into GIMP 2 and scaled to resolution size. For this test, the resolutions tested were 200×200, 300×300, and 500×500. Resolution 100×100 produced a poor quality map and was not on the canvas; thus testing for 100×100 resolution was eliminated. A map of resolution 1000×1000 created an excellent map, but this resolution was too memory intensive for the A* package and would core dump. The x and y positions for each of the points were marked as in the previous steps. The two points are shown in the figures below and are denoted for each of the resolutions. Figure 39 shows Point A and B along with their x and y coordinates for resolution 300×300.

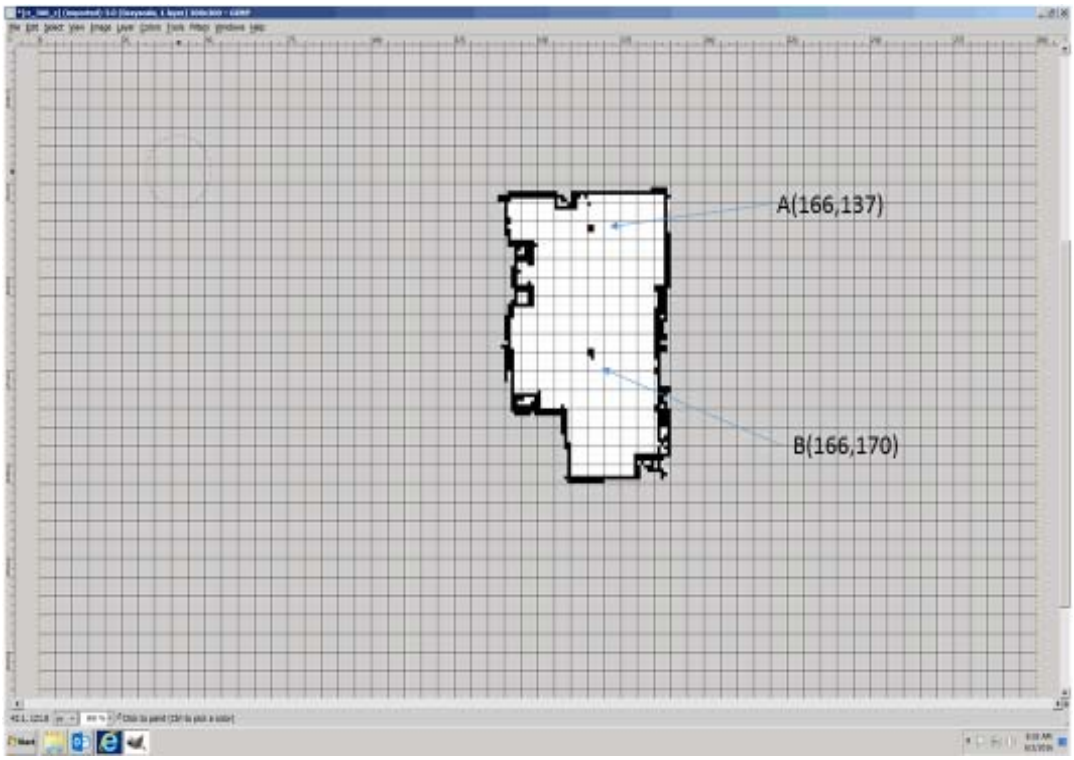

Figure 39. Gimp 300×300.

When running the A* algorithm, the start point A was provided by AMCL, and the destination Point B was provided from the Gimp software. The room was larger than in Test Site One in Table 12-14. The distance from where the robot ended and the position of Point B in the physical map is shown in the Distance column, and the time to travel the path is in the time columns. In the 200×200 resolution map, Table 12, the error was from 48 to 70 inches, with an average of 61 inches for the five runs. In the 300×300 resolution map, Table 13, the error was from 35 to 80 inches, with an average of 55.4 inches for the five runs. In the 500×500 resolution map, Table 14,the error was from 56 to 108 inches, with an average of 69.6 inches for the five runs. Overall, for Test Site 3, the 300×300 resolution performed best.

Table 12. Results for 200×200 on Test Site Three.

	Generated Map		AMCL		
	A	B	A	Time (sec.)	Distance (in.)
Run 1	(116,95)	(107,128)	(110,110)	26.34	60
Run 2	(116,95)	(107,128)	(110,110)	25.89	63
Run 3	(116,95)	(107,128)	(110,110)	26.05	64
Run 4	(116,95)	(107,128)	(120,110)	13.79	48
Run 5	(116,95)	(107,128)	(110,110)	27.75	70

Table 13. Results for 300×300 Test Site Three.

	Generated Map		AMCL		
	A	B	A	Time (sec.)	Distance (in.)
Run 1	(166,137)	(166,170)	(160,150)	29.62	36
Run 2	(166,137)	(166,170)	(160,150)	38.93	60
Run 3	(166,137)	(166,170)	(160,150)	29.08	35
Run 4	(166,137)	(166,170)	(170,150)	30.83	66
Run 5	(166,137)	(166,170)	(170,170)	32.17	80

Table 14. Results for 500×500 Test Site Three.

	Generated Map		AMCL		
	A	B	A	Time (sec.)	Distance (in.)
Run 1	(269,248)	(248,276)	(250,250)	41.91	108
Run 2	(269,248)	(248,276)	(260,260)	26.31	62
Run 3	(269,248)	(248,276)	(260,260)	23.12	56
Run 4	(269,248)	(248,276)	(260,250)	40.56	62
Run 5	(269,248)	(248,276)	(260,250)	36.51	60

6.6. Conclusion

This research shows that the resolution of the map has as effect on the robot's localization and path finding ability. The Explorer performed best at 300×300 resolution in all cases.

The research in Chapter 6 provides the ERDC with an increased level of autonomy from zero or tethered to autonomous level three, when searching unknown environments. When this research began, the ERDC was using a tethered robot to explore unknown areas. We have tested and selected the SLAM algorithm via simulation, performed physical testing on a mobile robot and concluded that HectorSLAM is an excellent choice for mapping an unknown area. Also, a ROS package AMCL, was implemented, which allows the robot to localize itself to the SLAM generated map. Once localized to the map, the mobile robot was able to traverse a path generated with the A* algorithm.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

The main goals of this dissertation were to enhance a commercial off the shelf (COTS) robotic platform with the ability to generate a map of an unknown area, to localize the robot to the generated map, and to increase the level of autonomy from tethered to a level of five or more. The UGV was tested with three SLAM algorithms, CORESLAM, Hector SLAM, and gMapping, in a simulated environment and a physical environment. Hector SLAM proved to be the superior SLAM algorithm, outperforming both gMapping and CORESLAM in both simulation testing and physical testing. It is my belief that Hector SLAM generated the best map because it requires only a laser scan, unlike gMapping and CORESLAM.

This research had a series of limitations: (1) ROS has a steep learning curve and there are unlimited packages for use, (2) ROS operates in a Linux environment, (3) lack of a dedicated test area that caused multiple moves and setups as time allowed, and (4) the UGV and laptop's WIFI signals proved unacceptable for the transmission of data. This WIFI issue was overcome by adding a radio to both the laptop and UVG and setting up a network between the two. Large amounts of data were then transmitted between the two.

7.2. Future Work

Further enhancements to the UGV include adding other plug-and-play devices such as Microsoft Kinect Sensor or the 360-degree 3-D sensor by DFROBOT. Each of these sensors will

produce a 3-D maps of the navigated area that can be overlaid onto the 2-D map. The Microsoft Kinect sensor costs about $100.00 and is easily installed. The Kinect consists of several sensors including a RBG sensor, a 3-D depth sensor, multi-array microphone, and an accelerometer (https://developer.microsoft.com/en-us/windows/kinect/hardware). The 360-degree 3-D sensor by DFROBOT is also inexpensive and should be easily programmed.

Increasing the level of autonomy from around three, to between six and ten is needed. Testing in a tunnel environment would show more of the strengths and weaknesses of this UGV.

REFERENCES

Angeli, A., Doncieux, S., Meyer, J. A., and Filliat, D. (2008, 22-26 Sept. 2008). *Incremental vision-based topological SLAM.* Paper presented at the Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference.

Banks, J., and Carson, J. S. (1984). *Discrete event system simulation*.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). *Monte carlo localization for mobile robots*.

Doray, E., Clymer, A., McKenna, J., Horner, D., McKenna, M., Anderson, T., et al. (2009, 11-12 May 2009). *Unmanned Tunnel Exploitation.* Paper presented at the Technologies for Homeland Security, 2009. HST '09. IEEE Conference.

Edwards, D., Bean, T., Odell, D., and Anderson, M. (2004). *A leader-follower algorithm for multiple AUV formations*: IEEE.

Fox, D., Burgard, W., and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research, 11*(3), 391-427.

Gage, D. W. (1995) UGV HISTORY 101: A Brief History of.

Kinney, P., Dooner, M., Nagel, J., and Trepagnier, P. (2006). *Kat-5: Systems based on a successful paradigm for the development of autonomous ground vehicles*.

Kohlbrecher, S. ((n.d.)). *Hector_mapping*. Retrieved from http://wiki.ros.org/hector_mapping.

Kramer, J. A. (2010). *Accurate localization given uncertain sensors.* University of South Florida.

MathWorks. ((n.d.)). Retrieved from http://www.mathworks.com

Morgan Quigley, B. G., Ken Conleyy, Josh Fausty, Tully Footey, Jeremy Leibsz, E. B., Rob Wheelery, and Andrew Ng. (2009). ROS: an open-source Robot Operating System.

Murphy, R. R. (2004). Rescue robotics for homeland security. *Communications of the ACM, 47*(3), 66-68.

Reynolds, J. (2005). An Exploration of Mapping Algorithms for Autonomous Robotic Mapping. *Colorado State University*.

Riisgaard, S., and Blas, M. R. (2004). SLAM for Dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*.

Rozman, J. (2009). Grid-based map making using particle filters. *International Journal of Autonomic Computing, 1*(2), 211-221.

Russell, S., and Norvig, P., (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*.

Santos, J. (n.d.). *Gmapping*. Retrieved from http://wiki.ros.org/gmapping.

Singer, P. W. (2008). Robots at war: the new battlefield. *Wilson Quarterly*, 30-48.

Song, I. (2002). Probabilistic Localization Methods for a Mobile Robot Navigation.

Stanculescu, A., and Sojka, M. (2008). Evaluation of the Monte Carlo Localization algorithm.

Staranowicz, A., and G. Mariottini. 2011. “Robotic Simulation Guide.” ASTRA Robotics Lab, Department of Computer Science and Engineering, University of Texas, Arlington, Texas. Accessed April 23, 2015. http://ranger.uta.edu.

Thrun, S. (2002b). Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1–35.

Thrun, S. (2006). The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research, 25*(5-6), 403-429.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2001). Robust Monte Carlo localization for mobile robots. *Artificial Intelligence, 128*(1-2), 99-141.

University of Tennessee Knoxville. (2007). Player/Stage: Getting Started Guide. Retrieved from University of Tennessee Knoxville, Machine Language website.

Voth, D. (2004). A new generation of military robots. *Intelligent Systems, IEEE, 19*(4), 2-3.

Welch, G., and Bishop, G. (1995). An introduction to the Kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC, 7*(1).

APPENDIX A

Estimate

CoroWare
1410 Market Street
Kirkland, WA 98033
Phone: (800) 641-2676, option 1

DATE: February 15, 2011
ESTIMATE: EXPLORER - USAE

TERMS: Net 30

BILL TO:
U.S. Army Corps of Engineers
Attn: Doris Turnage

SHIP TO:
U.S. Army Corps of Engineers
Attn: Doris Turnage

DESCRIPTION	Quantity	Amount	Total
Explorer Mobile Robot	1	$ 8,499.00	$ 8,499.00
Power Options : Field "Hot Swap", Lithium Ferrous Phosphate (LiFePO)	1	$ -	$ -
Sensors: GPS Receiver, 2 MegaPixel Color WebCam	1	$ -	$ -
Mainboard Options: CUDA-capable	1	$ 600.00	$ 600.00
Operating System: Ubuntu Linux	1	$ 100.00	$ 100.00
Robotics Software Platform : Robot Operating System (ROS)	1	$ 200.00	$ 200.00
Storage media: 60 Gigabyte solid-state drive	1	$ 300.00	$ 300.00
Accessories: Pan/tilt camera	1	$ 750.00	$ 750.00
Laser Range Finder: Outdoor LRF UTM-30LX	1	$ 6,999.00	$ 6,999.00
Battery Pack (Lithium Ferrous Phosphate (LiFePO); estimated 6-10 hours operation)	1	$ 700.00	$ 700.00
Intelligent Battery Charger and Conditioner	1	$ 200.00	$ 200.00
Mainboard Options: CUDA-capable	1	$ 600.00	$ 600.00
Subtotal			$ 18,948.00
Shipping (Within USA)	1	$ 350.00	$ 350.00
		SUBTOTAL	$ 19,298.00
		TAX RATE	0.00%
		SALES TAX	$ -
		OTHER	
		TOTAL	**$ 19,298.00**

WIRE TRANSFER INFORMATION:

Wells Fargo Bank
Account Owner: CoroWare Technologies Inc
Account Number: 1608861744
SWIFT Code: WFBIUS6S

This quote expires 90 days after the date of issue.

VITA

DORIS M. TURNAGE

P. O. Box 822883 ● Vicksburg, MS 39182 ● (601) 613-9785 ● reeturn@yahoo.com

EDUCATION

Ph.D., Engineering Science**,** University Of Mississippi, August 2016
Concentration: Computer Science
Dissertation Title: "Localization and Mapping Unknown Locations and Tunnels with Unmanned Ground Vehicles"

M.S., Computer Science, Mississippi State University, May 2001
Thesis: Vehicle Database Management System

B.S., Mathematics, University of Southern Mississippi, May 1984

PROFESSIONAL EXPERIENCE

Computer Scientist, August 1990-present
Department of Army
Engineer Research and Development Center
Vicksburg, MS 39180

Mathematician, May 1985 - August 1990
Department of Army
White Sands Missile Range
White Sands Missile Range, NM

PRESENTATIONS

Turnage, Doris (July 2011) Software Design of an Unmanned Ground Vehicle for Exploration of Complex Environments, National Defense Industrial Association Robotics Capabilities Conference, San Diego, California

Turnage, Doris (July 2009) Terrain Reasoning for Course of Action (COA) Development for a Medical Treatment Facility, ESRI Conference, San Diego, California

Turnage, Doris (July 2008) Terrain Reasoning for Course of Action (COA) Development for a Casualty Collection Point, ESRI Conference, San Diego, California