NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTONOMOUS INTERIOR MAPPING ROBOT UTILIZING LIDAR LOCALIZATION AND MAPPING

by

Jameson S. Payne

September 2020

Thesis Advisor:	Xiaoping Yun
Second Reader:	James Calusdian

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

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE AUTONOMOUS INTERIOR MAPPING ROBOT UTILIZING LIDAR LOCALIZATION AND MAPPING	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jameson S. Payne	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Combat actions are planned based on the best available information. In nearly all situations, significant uncertainty about the combat environment exists. This uncertainty contributes largely to friendly and non-combatant casualties. At the tactical level, operators are often required to enter hostile-occupied buildings without knowledge of the building layout. Military operators have begun to use robots to assist in missions of this type. In general, currently fielded robots lack autonomy and the ability to disseminate an accurate map, on-site, in real time. The purpose of this thesis is to examine the feasibility of an autonomous robot that can localize and build accurate 3D maps using only light detection and ranging (LIDAR). To accomplish this, a robot equipped with only LIDAR and a control algorithm for LIDAR localization and mapping (LLAM) were developed. Trials were then developed to determine if LLAM is a feasible model for interior 3D mapping. Navigation was accomplished using a potential field model adapted from previous work combined with the Hybrid A* search algorithm. Mapping and localization were conducted using the iterative closest point and normal distribution transform methods of point cloud registration. Experimentation revealed that LLAM is a feasible method for interior 3D mapping in real time. Further development of the algorithm may make fielding a LIDAR-equipped mapping robot possible with current mobile computing technology.

14. SUBJECT TERMS autonomous, robotics, light detection and ranging, LIDAR, LIDAR localization and mapping, LLAM	**15. NUMBER OF PAGES** 137
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AUTONOMOUS INTERIOR MAPPING ROBOT UTILIZING LIDAR LOCALIZATION AND MAPPING

Jameson S. Payne
Major, United States Marine Corps
BSME, Washington State University, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: Xiaoping Yun
Advisor

James Calusdian
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Combat actions are planned based on the best available information. In nearly all situations, significant uncertainty about the combat environment exists. This uncertainty contributes largely to friendly and non-combatant casualties. At the tactical level, operators are often required to enter hostile-occupied buildings without knowledge of the building layout. Military operators have begun to use robots to assist in missions of this type. In general, currently fielded robots lack autonomy and the ability to disseminate an accurate map, on-site, in real time. The purpose of this thesis is to examine the feasibility of an autonomous robot that can localize and build accurate 3D maps using only light detection and ranging (LIDAR). To accomplish this, a robot equipped with only LIDAR and a control algorithm for LIDAR localization and mapping (LLAM) were developed. Trials were then developed to determine if LLAM is a feasible model for interior 3D mapping. Navigation was accomplished using a potential field model adapted from previous work combined with the Hybrid A* search algorithm. Mapping and localization were conducted using the iterative closest point and normal distribution transform methods of point cloud registration. Experimentation revealed that LLAM is a feasible method for interior 3D mapping in real time. Further development of the algorithm may make fielding a LIDAR-equipped mapping robot possible with current mobile computing technology.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CPD	Coherent Point Drift
DoF	Degree of Freedom
FoV	Field of View
GNSS	Global Navigation Satellite System
GNSS/INS	GNSS-Inertial Navigation System
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
LLAM	LIDAR Localization and Mapping
LOAM	LIDAR Odometry and Mapping
NDT	Normal Distribution Transform
SDK	Software Development Kit
ROS	Robot Operating System
SOF	Special Operations Forces
ToF	Time-of-Flight

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Xiaoping Yun and Dr. James Calusdian. Dr. Yun, for your exceptional instruction throughout my graduate-level courses, which is the reason I became interested in this topic. Furthermore, your guidance during research has been incredibly valuable. You granted me the latitude to explore the subject as I saw fit and the guidance required to achieve something useful. Dr. Calusdian, I cannot begin to enumerate all the problems you helped me solve, or the ideas you provided. Your patience and genuine concern for the education of your students was apparent from the first course I took at NPS. I am grateful to both of you for your hard work and outstanding instruction.

I would also like to thank my family. To my wife, Jessica, you are everything to me. You are the reason I chose to pursue engineering after a decade in combat arms, a decision that I am grateful for every day. You have stood by me through the weekends of math and studying, and provided more support than you know. You are a wonderful wife and mother, and I love you very much.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The use of light detection and ranging (LIDAR) for mapping and obstacle avoidance in autonomous ground vehicles is an active area of study. The accuracy, precision, responsiveness, and continuously decreasing cost of LIDAR makes it attractive as a solution for creating detailed, highly accurate maps. The use of LIDAR in autonomous localization is a newer, less developed area of study but shows promise for the future of autonomous vehicles.

Recently, commercially available LIDAR systems have significantly dropped in cost and increased in resolution. Velodyne, for example, produces a commercially available LIDAR that can produce as many as 4.8 million 3D points per second with return ranges up to 245 m [1]. This level of resolution can produce highly detailed 3D models; however, computer processing of a point cloud that large can create significant problems. The overwhelming amount of data generated by 3D LIDAR systems creates a bottleneck that makes localization and mapping in real time challenging. LIDAR-based localization and mapping has a wide range of uses today. In most applications, LIDAR is primarily used to build a map. LIDAR-based localization is typically combined with other sensors to estimate the position and orientation, or pose, of a robot. Frequently, LIDAR-based localization and mapping is done in post-collection processing. During collection of LIDAR data, the robot or autonomous vehicle is localized by a suite of other sensors. After it has run its intended trajectory, the data is then processed to create a three-dimensional map. This model works well for many industrial applications; however, it requires larger, heavier, more complex robots that are not well suited for military ground reconnaissance purposes. This thesis examines the applicability of LIDAR-based localization and mapping to military operations, the motivation, related work, and purpose of this research are outlined in this chapter.

A. MOTIVATION

Infantrymen and special operators are often required to enter hostile buildings with no prior knowledge of the building layout. Typically, hostile actors inside the building know the layout and thus have a tactical advantage. The author, as an infantry officer and expeditionary ground reconnaissance officer has encountered this problem firsthand without a suitable solution available. Recently, infantry, SOF, and reconnaissance units have fielded various robots such as the Recon Scout XT depicted in Figure 1. These robots are controlled by a human that receives feedback from an onboard camera to the handheld remote control. The video is valuable in identifying hostile actors and occasionally booby traps, but there is no residual map. Frequently these recon robots need to be employed when a small unit is tactically distributed in such a manner that only the operator and perhaps the unit commander can view the video feed. Communicating the layout of a building by voice can be difficult and ineffective. As small tablet devices become more widely proliferated among tactical operators, ideally, a robot could build a map and transmit it instantly to all operators on-scene for increased situational awareness.

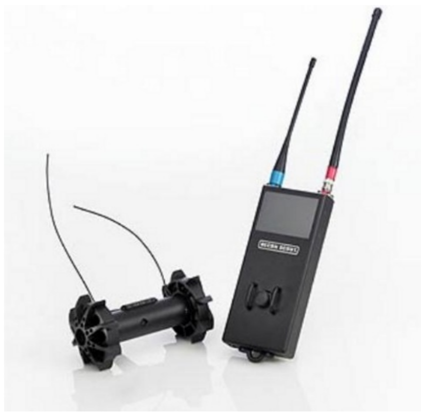


Figure 1. ReconRobotics' Recon Scout XT Throwable Robot. Source: [2].

As the field of autonomous robotics continues to advance, a technological solution to this problem is becoming more feasible. LIDAR is widely used in remote sensing, autonomous driving, and robotics. However, further research needs to be conducted into how this technology can be adapted to benefit the tactical-level warfighter. This work aims to examine whether LIDAR can be used effectively on a small mobile robot to improve situational awareness from a tactical perspective. Specifically, whether a robot can autonomously map the interior of rooms using only LIDAR without additional sensors, such as inertial measurement units, GPS, monocular or stereo cameras, wheel encoders, etc.

Utilizing a single LIDAR sensor for both localization and mapping is a relatively new approach to solving the simultaneous localization and mapping (SLAM) problem in autonomous robotics. Multiple sensors add weight and complexity to a robot. Tactical level operators need lightweight, simple, and resilient gear that will function in austere conditions. Additionally, the majority of research in the field of LIDAR localization and mapping (LLAM), sometimes called LIDAR odometry and mapping (LOAM), is primarily focused on increasing the resolution of a 3D map and reducing drift of the robot, usually with the trajectory of the robot being controlled by a human. Tactical-level operators do not need a highly detailed interior map; a quickly developed map with limited resolution that can be built on-scene autonomously would be more beneficial. This research seeks to determine whether current technology is sufficient to develop that capability.

B. RELATED WORK

This research falls under a larger autonomous robotics project conducted within the Electrical and Computer Engineering Department at the Naval Postgraduate School. The goal of this larger umbrella project is to create a robot capable of autonomous navigation from any location on campus to any other location on campus regardless of the terrain, structures, or obstacles encountered. Hagardine [3] developed algorithms to avoid both static and moving obstacles using a single 2D LIDAR in an unstructured outdoor environment. Similarly, in [4], Miyakawa used downward-looking LIDAR to avoid low-profile obstacles and a sensor suite including optical flow sensors and GNSS/INS to localize the robot. Lebrun [5] expanded on the work of Hagardine by integrating visual classification of obstacles from camera data and the random forest machine-learning algorithm. Finally, Magee [6] examined autonomous navigation through convolutional neural network image classification. In her work, she used a SLAM map collected from 2D LIDAR for waypoint generation. All of these theses contributed to this research to some degree, much of these previous works have been adapted for the purpose of LLAM, particularly obstacle detection and avoidance algorithms.

Outside of Naval Postgraduate School, significant work has been done in both academia and industry on the use of LIDAR in autonomous robotics. Moosmann and Stiller [7] developed an LLAM model that was effective in building a 3D map despite high sensor noise. In [8], Zhang and Singh introduced a real-time mapping and localization algorithm using at 2D Hokuyo UTM-30LX laser scanner, which was mounted to a motor that rotated the entire LIDAR unit about an axis orthogonal to the laser scan axis of rotation. This allowed them to generate a 3D point cloud from a 2D LIDAR. This method was interesting but complicated due to the constant two-axis rotation, which can cause poor point cloud registration. Ji et al. [9] introduced loop-closure correction to LOAM based on point cloud segmentation and random forest algorithm classification; this represented a complete LLAM solution to solve the SLAM problem. Loop-closure is a necessary step for large-scale or highly accurate mapping; however, as will be discussed later, it may not be necessary for tactical level operations.

Lightweight and ground-optimized LOAM was introduced in [10] for ground vehicles on variable terrain. This framework introduced the novel concept of segmenting a point cloud and obtaining one part of the required transformations from planar features, and another part from edge features. This allowed for a six degrees of freedom (DoF) model while reducing the workload for the robot onboard computer. In [11], Lin and Zhang develop a method for scan matching for limited field of view (FoV) LIDARs with non-repetitive scanning patterns. All of these works introduced novel concepts, but in all cases, the LIDAR system was mounted to a vehicle or robot with the trajectory being pre-planned or controlled remotely by a human, or in some cases, such as in [11], actually attached to a computer carried by a human. These works did not solve the tactical on-site reconnaissance problem addressed here. Additionally, although they successfully demonstrated that LLAM can be carried out accurately in real-time, to include loop-closure based drift correction, they did not seek to conduct mapping of an unknown area fully autonomously. In order to operate autonomously, the robot must be able to localize and map, in addition to numerous other processes including obstacle detection and avoidance, incrementally re-planned exploration, waypoint planning, and route optimization. All of these processes require additional computational power on a mobile robot, which may exceed the limits of current mobile computing technology.

C. PURPOSE AND GOAL

The purpose of this research is to examine the feasibility of a LIDAR-only based autonomous mobile robot and its capacity for scanning rooms in a tactical environment. Most of industry and academia are currently focused on the development of highly accurate, detailed, large-scale, and semi-autonomous LLAM capability. This research, on the other hand, focuses on the development of a rapid, efficient, limited-scale, fully-autonomous capability. This includes examining not only the methods of LIDAR scan matching, route planning, obstacle avoidance, and frontier exploration, but also their integration. It also requires an examination of the map resolution that is required for tactical level planning and the most effective method of visually displaying the 3D interior of rooms.

The goal is to construct a robot that can successfully map the interior of rooms autonomously and in real time. For this project, a ground vehicle was used for experimentation. However, as the future of this technology likely involves an aerial robot, such as a LIDAR-equipped drone, the goal included development of a six DoF model that could be easily adapted to an aerial robot.

The rest of this thesis is organized as follows. The hardware and software used for the construction of two LLAM robots is discussed in Chapter II. The theory of LLAM, methods of LIDAR scan matching, and map-making techniques are examined in Chapter III. The design of the robots and the control algorithms implemented in this work are detailed in Chapter IV. Results of experimentation on these robots are examined in Chapter V and conclusions on the feasibility of LLAM for tactical applications are discussed in Chapter VI.

THIS PAGE INTENTIONALLY LEFT BLANK

II. HARDWARE AND SOFTWARE DESCRIPTION

A. HARDWARE

The hardware used in this thesis research consisted of an indoor two-wheeled mobile robot, a 2D LIDAR system, a 3D time-of-flight (ToF) camera, an onboard computer, and a laptop or desktop computer. This robot was used previously in [6]; in this work it was adapted to carry a 3D ToF camera.

1. Omron Adept MobileRobots Pioneer 3-DX (P3-DX)

The base robot used was an Omron Adept MobileRobots Pioneer 3-DX, as seen in Figure 2. It is a two-wheeled robot with a third caster, allowing it to remain balanced. According to [12], it operates with a two-motor differential drive and encoders with 33,500 counts/rotation each. It also has a 16-sensor sonar array, as well as front and rear bumpers. The P3-DX has a maximum translational speed of 1.4 m/s and a maximum rotational speed of 300 deg/s, but in experimentation, those speeds were limited to no more than 0.4 m/s and 40 deg/s for safety and to ensure that the robot did not outpace the LLAM algorithm. The robot has a minimum turn radius of zero m, and a swing radius of 26.7 cm, which is well suited for indoor exploration. However, with a ground clearance of only six cm, it cannot overcome many obstacles that it cannot circumvent [12].

In this work, the P3-DX was modified to carry a Hokuyo UTM-30LX scanning laser range finder (2D LIDAR), PMD Pico monstar 3D ToF Camera, and a SlimPRO SP675SP Mini PC or Dell Latitude 3560 Laptop.

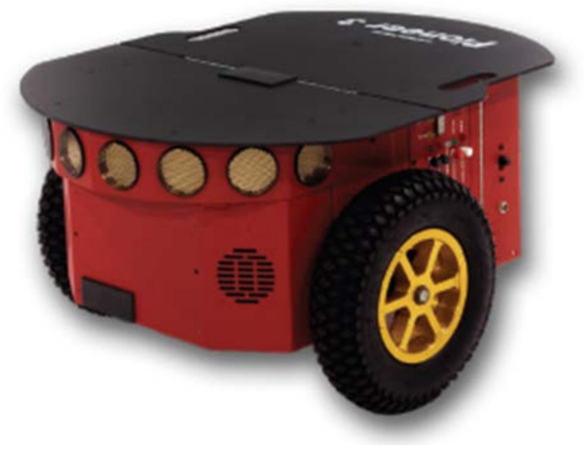


Figure 2. Omron Adept MobileRobots Pioneer 3-DX. Source: [13]

2. PMD Technologies CamBoard Pico monstar

The Pico monstar, shown in Figure 3, is a 3D Time-of-Flight (ToF) camera also known as a scanner-less LIDAR. According to [14], it operates with four 805 nm modulated IR lasers, which are transmitted from the camera, reflected off surfaces, and received by a 3D imager that measures the phase shift between the transmitted and received laser pulse. Then the sensor firmware calculates the distance to the point of reflection. This system has a maximum range of approximately 6 m and a field of view of 100° × 85° corresponding to a 352 × 287 (≈100,000 pixels) image. Each frame returned includes XYZ cartesian coordinates in meters, an integer grayscale value (0 to 255), which indicates the intensity of the laser return, and an integer depth confidence value (0 to 255). In this thesis, only the XYZ coordinates and the depth confidence value were used; however, the grayscale value does provide the ability to do image analysis on a flat grayscale image of a scene and could prove useful in future research.

Figure 3. PMD Technologies PMD Pico monstar 3D Camera. Source: [14].

The system is capable of capturing up to 60 fps; however, increased framerate reduces range significantly, and in this work, the maximum framerate was limited to 10 fps. The depth resolution of the camera varies both with framerate and with distance. According to the manufacturer, it is less than or equal to 1% of the distance from 1–6 m at 5 fps [14]. Conveniently, this camera can be powered entirely from the USB 3.0 connection and does not require an additional power source. For experimentation, the Pico monstar was mounted on the front of the robot for obstacle avoidance and to build a 3D interior map. The limited field of view (FoV) and range were not ideal for interior mapping, and thus it was combined with the Hokuyo 2D LIDAR, which was used primarily for navigation.

3. Hokuyo UTM-30LX

The Hokuyo UTM-30LX seen in Figure 4 is a 2D laser range scanner with an FoV of 270° and a range of approximately 30 m, but in some conditions up to 60 m. It operates with a spinning IR laser at a wavelength of 905 nm. With an angular resolution of 0.25°, the scanner can return 1080 points per scan, and with its extended range, it frequently does return 1080 useable points in an indoor environment. The accuracy of the UTM-30LX is ±30 mm up to 10 m, and ±50 mm between 10 m and 30 m. This LIDAR can return scans at 25 ms/scan or 40 Hz [15].

Figure 4. Hokuyo UTM-30LX Scanning Laser Range Finder. Source: [15].

The FoV and range of the Pico monstar is limited. In even moderately sized rooms, such as the ECE Control Systems Laboratory, the robot could travel sufficiently far from the walls or workbenches that the Pico monstar would not return enough valid points in each scan. This led to scan matching degeneration and ultimately a loss of localization, further discussed in Chapter V. The long range, relative accuracy, and high scan rate of the Hokuyo made it a good solution in the absence of a 3D spinning LIDAR. During experimentation, the Hokuyo was used in tandem with the Pico monstar, with the Hokuyo collecting accurate range and angle data for pose estimation, and the Pico monstar collecting 3D data for more robust obstacle avoidance and to map in 3D.

4. Dell Latitude 3050 Laptop

For experimentation purposes, two robots were constructed. The first robot (Robot 1) used a Dell Latitude 3050 laptop as an on-board processing unit. The Dell laptop contained an Intel i3-5005U 2.00 GHz CPU, and 16 GB RAM. It ran Windows 10 and MATLAB 2019b. During experimentation with Robot 1, the Dell Laptop was mounted on top of the robot and processed all localization, mapping, navigation, and obstacle avoidance algorithms on board. This laptop was powered by its internal battery.

5. SlimPRO SP675P i7 Mini PC

The second robot constructed (Robot 2) had identical hardware to the first robot with one exception; the Dell laptop was replaced with a SlimPRO SP675P Mini PC. According to the manufacturer, it has a 3rd generation Intel i7 Core mobile CPU, 8GB RAM, and has dimensions of 5.75”(W) × 10.0”(D) × 1.65”(H). The SlimPRO has one Gigabit LAN port, four USB 3.0 ports, and two USB 2.0 ports to connect sensors and other peripheral devices. It runs on 12V/60W DC power supplied from the robot battery [32]. The SlimPRO operated with Linux Ubuntu 18.04.4 and Robot Operating System (ROS) Melodic. The SlimPRO computer was used for retrieving LIDAR data from the Hokuyo and PMD Pico monstar, processing 2D laser scan transformations, obstacle avoidance, and navigation. Due to its limited computational power, 3D point cloud registration and 3D map building were performed on a Dell desktop computer.

6. Dell Optiplex 7040 Desktop

The SlimPRO on Robot 2 communicated with a Dell Optiplex 7040 Desktop computer running the 3D point cloud registration and 3D map building algorithms. This communication with the SlimPRO was conducted wirelessly using ROS and 802.11 protocol. This computer hardware included a 3.2 GHz Intel Core i5-6500 CPU, and 8 GB RAM. It ran Windows 10 and MATLAB 2019b. The desktop received both 2D pose information and 3D LIDAR data and registered that data for map building in 3D. More powerful miniature computers exist that could likely handle 3D point cloud processing onboard the robot; however, this architecture had several advantages. First, it split the computational cost of autonomous interior mapping between two computers, speeding up the robot. More importantly, from a tactical perspective, an autonomous mapping robot could be captured or disabled by the enemy while mapping. If map data were stored only on the robot, no map would exist for the operators. Transmitting data and building the map in parallel on another machine allows the operator to get a partial map of an interior space, even if the robot is disabled prematurely.

B. SOFTWARE

A wide variety of software exists for the control of autonomous robots. Each programming language and software interface has its own benefits and shortfalls. This work was primarily done in MATLAB for several reasons. First, previous ECE thesis work described in Chapter I was coded in MATLAB; significant pieces of that work could be easily adapted for this thesis without being reinvented. Second, MATLAB includes packages and toolboxes that support many of the functions required in a robust fashion, without the need to develop these from scratch. These qualities make MATLAB ideal for prototyping and development especially if it is an individual or small group project such as a thesis. Additional software used in this research include, PMD Royale software and ROS, which are detailed in subsequent sections.

1. MATLAB

For this work, the algorithms developed for robot control were coded in MathWorks MATLAB Releases 2019b and 2020a. MATLAB is a programming language built for scientists and engineers using a matrix-based method of computational mathematics [17]. MATLAB includes many packages and toolboxes that have robustly built functions to simplify programming. This work makes use of the Robotics Systems, ROS, Computer Vision, Image Processing, Navigation, and Mapping Toolboxes. All MATLAB scripts for this thesis are included in the appendices. MATLAB also includes the ability to run ROS services and nodes from computers running a Windows operating system. ROS itself is written for Linux. However, utilizing the ROS toolbox in MATLAB allowed the robot mapping algorithm to split the computational workload between the onboard computer (Linux) and another Windows-based computer, already discussed.

2. Royale Software Suite

The Royale Software Suite is a software development kit (SDK) that is included with the purchase of a Pico monstar 3D camera. The Royale software contains a tool to visualize 3D data called the Royale Viewer (version 3.23.0.86 was used). It also contains tools to support interactivity with the camera through C++, C, Python, OpenCV, OpenNI2, MATLAB, ROS, and DotNet [18]. In this work, the MATLAB wrapper was used to interact with the camera in the MATLAB environment. This wrapper contained the necessary dynamic link libraries (*.dll), as well as MATLAB class definitions and functions to use the camera in the MATLAB environment. It contained functions to set parameters for the camera, trigger it to capture, and retrieve data. Parameters that can be set for the camera include frame rate, exposure mode, exposure time, trigger mode, filter levels, etc. [19]. The camera comes with nine preset modes for different applications, such as hand gesture recognition, 3D object scanning, or room scanning [19].

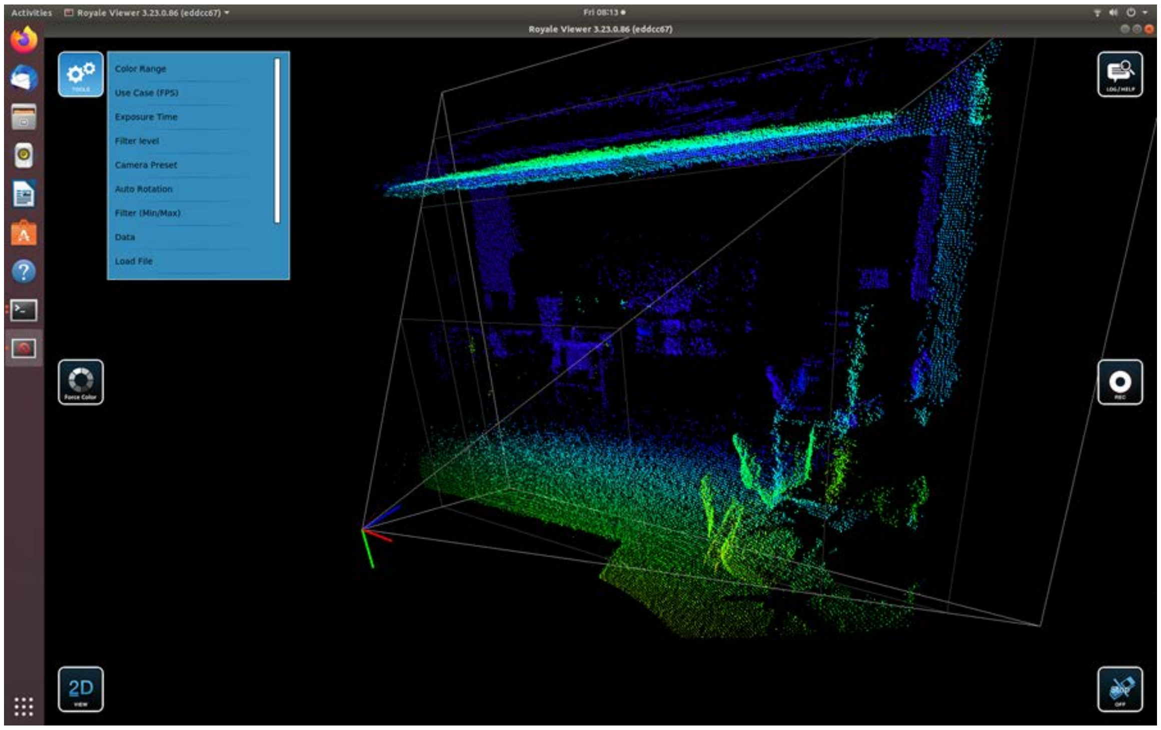


Figure 5. Royale Viewer Point Cloud Visualization Software

The Royale Viewer, as seen in Figure 5, allows the user to test the preset modes and adjust the majority of camera parameters using slider bars. This is particularly useful for fine tuning camera settings that correspond to a specific application or environment and was used to test a number of different camera configurations. When the right camera parameters are found, those values can then be coded into the MATLAB environment to be set each time the camera is initialized. Additionally, the Royale Viewer allows the user to record a 3D stream from the camera in a single file.

3. Robot Operating System (ROS)

ROS is an open-source framework for programming robots built in Linux environments. It includes various tools and packages that can simplify the complex task of controlling robots [24]. In this research, ROS Melodic Morenia was used to establish a network from the Linux Ubuntu 18.04.4 SlimPRO PC. This ROS network interfaced with MATLAB and allowed for the combination of data retrieved from the 2D and 3D LIDAR sensors. The only ROS package used that was not included in the MATLAB ROS toolbox was the *urg_node* package for communicating with the Hokuyo LIDAR. The ROS package built for communication with the PMD Pico monstar has not been maintained for newer distributions of Ubuntu or ROS, thus that sensor was interfaced with MATLAB directly using the Royale SDK.

III. LOCALIZATION AND MAPPING

Simultaneous localization and mapping (SLAM) is a process by which a robot maps the surrounding environment using various types of sensors while simultaneously estimating the robot position within that environment [20]. Many methods exist for SLAM, including an increasing number based on LIDAR. LIDAR-based SLAM is essentially a scan-matching problem where laser scans, either 2D or 3D, are taken sequentially while the robot is in motion. These scans, often called point clouds, are then compared to one another in a process known as registration. Registering two sequential point clouds provides a geometrical transform that can be applied to one point cloud in order to make it match the other point cloud. This transform can be used to estimate the relative movement of the robot [20]. As new point clouds are received and are registered, the incoming point cloud is transformed by the accumulated transformations of the point cloud registrations before it and added to the map. Several methods exist for registration and mapping, and three of those methods will be discussed in this chapter.

A. REFERENCE FRAMES

The robot uses several reference frames for localization and mapping. Some of the reference frames are static and others move. Additionally, each of the sensors operates in a different reference frame, which must be transformed to a common reference frame to be integrated.

The primary fixed reference frame is the {WORLD} frame. The {WORLD} frame is a right-handed cartesian coordinate system with the X-direction forward from the rotational center of the robot. The origin of the {WORLD} frame is where the rotational center of the robot lies when it is initialized.

The primary moving reference is the {ROBOT} frame. It is a right-handed cartesian coordinate system with the origin at the center of the robot rotational axis (Z-axis) and the X-direction forward. When the robot is initialized, the {ROBOT} frame and {WORLD} frame are the same until the robot moves. Six DoF and three DoF {ROBOT} frames were considered in this work. Throughout experimentation, the origin of the {ROBOT} frame translated and rotated in the X-Y plane only because the P3-DX is a ground mobile robot not capable of negotiating stairs, and the interior of the test spaces was flat without ramps. In effect, all experimentation had three DoF. However, throughout this work when the {ROBOT} frame is labeled as three DoF, that means that it was coded in software to only consider three DoF, X-Y translation, and Z rotation (Yaw).

When the {ROBOT} frame is labeled six DoF, the LLAM code was written to consider translation in all three axial directions, as well as the other two rotational components (pitch and roll). This was done both to examine the increased computational cost of six DoF and also because follow-on research should attempt to apply LLAM to an aerial robot with six DoF.

The {PICO} frame of reference is a moving frame specific to the Pico monstar sensor. Point clouds returned from the Pico monstar are in a right-handed cartesian coordinate system with the Z-axis forward, Y-axis down, and X-axis right with respect to the camera lens when it is mounted upright. The origin being at the center of the camera lens.

The {HOKUYO} frame is a 2D cartesian frame of reference with the origin at the center of the Hokuyo sensor. The X-axis is forward, and the Y-axis is left. The rotation of the {HOKUYO} frame is fixed to the rotation of the {ROBOT} frame. The {HOKUYO} frame is essentially the X-Y plane of the {ROBOT} frame translated in the X and Z directions. The Hokuyo sensor returns laser readings in both polar and cartesian coordinates. Both formats are used throughout the robot programming. However, since the conversion between the two formats is simple and occurs in the local reference frame of the sensor, only the cartesian values are referenced in the {HOKUYO} frame for consistency.

In this work, the reference frame is denoted by a leading superscript before the variable. In the case of transformations between reference frames, the reference frame being transformed is denoted by a leading subscript. For example, the transformation T that maps a point x (XYZ coordinates) from the {ROBOT} frame to the {WORLD} frame of reference is denoted

$$ {}^{w}x = {}^{w}_{R}T\, {}^{R}x \,. \tag{1} $$

B. POINT CLOUD REGISTRATION

Many 3D models are built using 3D scanning processes, such as LIDAR; however, in many situations, it is only possible to collect a partial scan from any one perspective. In order to create a complete 3D model, the collection of partial scans must be combined into one scan [21]. This process is called registration and is applied to both point cloud and mesh-based 3D models. It typically involves three steps. First, correspondence is found between points in consecutive scans. Next, from these point correspondences, a transformation is calculated that will minimize the distance between corresponding points. Finally, transformed point clouds are merged into a single point cloud representing a complete model [21]. Numerous methods for point cloud registration exist, and many variations of each method have been developed to suit particular applications. In this thesis, we will examine three common methods of point cloud registration: Iterative Closest Point (ICP), Coherent Point Drift (CPD), and Normal Distribution Transform (NDT).

Registration returns either 2D or 3D transformations. Three parameters are used in 2D transformation: the translational parameters t_x and t_y and the rotational angle parameter ϕ. Collectively, the three parameters are written into a vector p as

$$ p = [t_x \quad t_y \quad \phi]\,. \tag{2} $$

Two-dimensional transformation is given by [22]

$$ T(p,x) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} x + \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \tag{3} $$

where x is the two-dimensional point being transformed.

In this work, all 2D transformations occurred in the {ROBOT} frame or the X-Y plane of the {WORLD} frame, and thus the rotational parameter ϕ is positive in the counter-clockwise direction.

Three-dimensional transformations are defined by a 3×3 rotation matrix, which is a combination of rotations about each reference axis, as well as a 3×1 translation vector. There are six parameters for a 3D transformation,

$$p = [t_x, t_y, t_z, \phi_x, \phi_y, \phi_z], \tag{4}$$

where the three values of *t* represent the translation in each dimension and the three values of ϕ represent the rotation around each reference axis. The 3D transformation is then given by [22]

$$T(p,x) = \begin{bmatrix} \mathrm{c}_y \mathrm{c}_z & -\mathrm{c}_y \mathrm{s}_z & \mathrm{s}_y \\ \mathrm{c}_x \mathrm{s}_z + \mathrm{s}_x \mathrm{s}_y \mathrm{c}_z & \mathrm{c}_x \mathrm{c}_z - \mathrm{s}_x \mathrm{s}_y \mathrm{s}_z & -\mathrm{s}_x \mathrm{c}_y \\ \mathrm{s}_x \mathrm{s}_z - \mathrm{c}_x \mathrm{s}_y \mathrm{c}_z & \mathrm{c}_x \mathrm{s}_y \mathrm{s}_z + \mathrm{s}_x \mathrm{c}_z & \mathrm{c}_x \mathrm{c}_y \end{bmatrix} x + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \tag{5}$$

where c represents cosϕ and s represents sinϕ, the subscript of c or s determining which value of ϕ to use.

Transformations can be rigid, non-rigid, or affine. A rigid transformation preserves the shape and size of objects in the scene, applying the same transformation to all points. An affine transformation allows for shearing and changes of scale, as well as the translation and rotation of points from a rigid transformation [27]. The Pico monstar has a short range and high capture speed. At the low speeds, such as in this work, there is little motion blur in each frame, and thus rigid transformations are preferred. Non-rigid transformations allow the shape of objects to change, points are transformed differently using a displacement field [27]. Non-rigid transformations were not used in this LLAM algorithm.

1. Iterative Closest Point Method

Iterative Closest Point is a popular approach to point cloud registration and used in many applications of LLAM including those in [8], [9],[10],[11], and others. ICP is based on two assumptions: first that consecutively scanned point clouds are of the same environment, and second that they are not too far apart [21]. The first assumption is fairly obvious as there would be little point in matching scans of two different environments. The second is not as intuitive but is important as it is often the cause of matching degeneration discussed later. The problem is essentially to estimate the rigid transformation that will map one point cloud to another. The ICP algorithm does this as follows [21,22]:

1. Given two consecutive sets of points in three-dimensional space.
2. For all the points in the first set, find the closest point in the second set.
3. Find the rigid transformation that minimizes the distance between the points in the first set and the corresponding points in the second set.
4. Apply the transformation to the points in the second set.
5. Iterate until convergence is found.

Convergence is typically considered to be reached when the closest point correspondences from step 2 do not change [21]. In [21], Bærentzen et al. provide a more in-depth explanation of how the rotational and translational components of the transformation are estimated. Several problems exist with the ICP algorithm. First, both point clouds may not have the same number of points, thus correspondences between points are not necessarily unique, and some points may remain unpaired [22]. Second, the assumption from initialization is that the closest point is the corresponding point, which may not be true. However, it is among the most popular algorithms because it still returns good results and is less computationally expensive than other methods [21, 22].

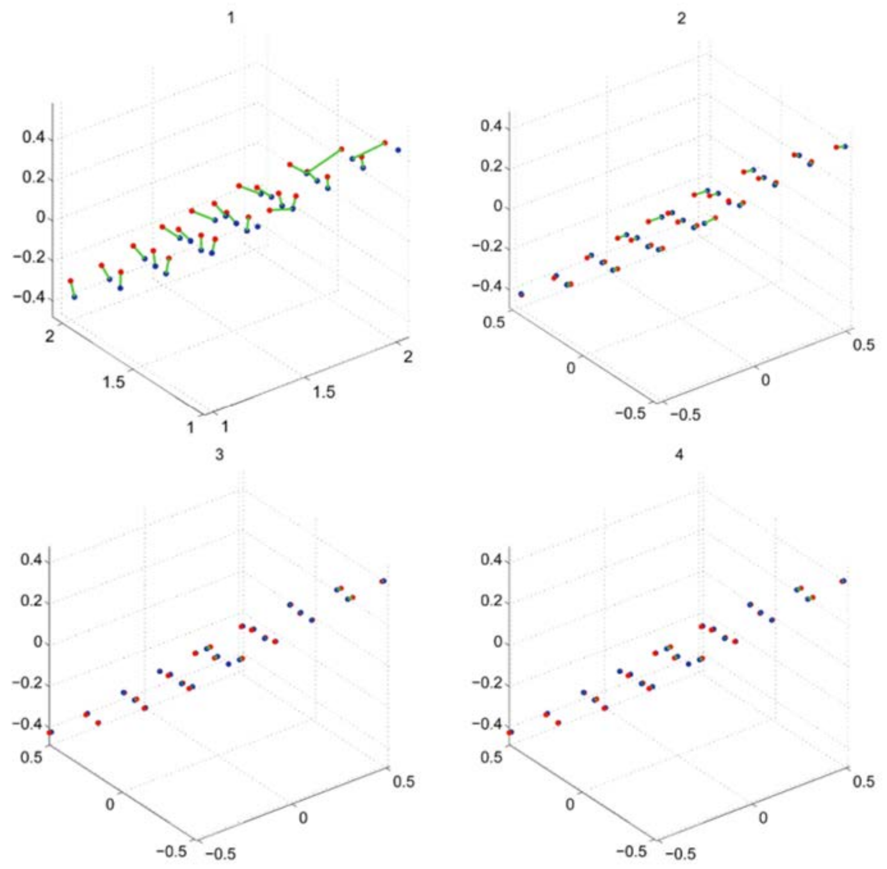


Figure 6. Four Iterations of The ICP Algorithm Aligning Red Points to Blue Points, the Correspondences Depicted as Green Lines. Source: [21].

A visual depiction of ICP is seen in Figure 6. In the first iteration, correspondence is found between the points in the red set and the closest points in the blue set. A transformation is calculated that maps the blue points closer to the red points. In the second iteration many of the point correspondences do not change, however, a few different correspondences are established. The transformation process is repeated. Examining the third and fourth iterations of the algorithm, it can be seen that none of the point correspondences change, this is typically considered convergence and terminates the algorithm [21].

Most ICP algorithms implement a k-dimensional tree (KD tree) searcher to improve the efficiency of finding point correspondences. A KD tree is a geometric data structure for organizing multi-dimensional points into a searchable tree. It is the most popular structure for searching ranges and nearest neighbors [23]. The KD tree begins with a root cell, which contains all the data in the set. It is then created by finding the median value along one dimension of the tree and splitting the point data into partitions with a hyperplane perpendicular to the axis being considered. Points on either side of the hyperplane are partitioned by the value of the considered dimension such that larger values fall on one side of the plane (right for example), smaller values on the other (left). Partitioning is repeated along the other dimensions creating new levels of the tree. When the last dimension is considered, the algorithm then returns to the first dimension and further partitions data within each of the existing partitions. This continues recursively until only single points remain in each partition [23]. The construct makes searching a KD tree for points within a specific range or nearest neighbors simple and efficient [23].

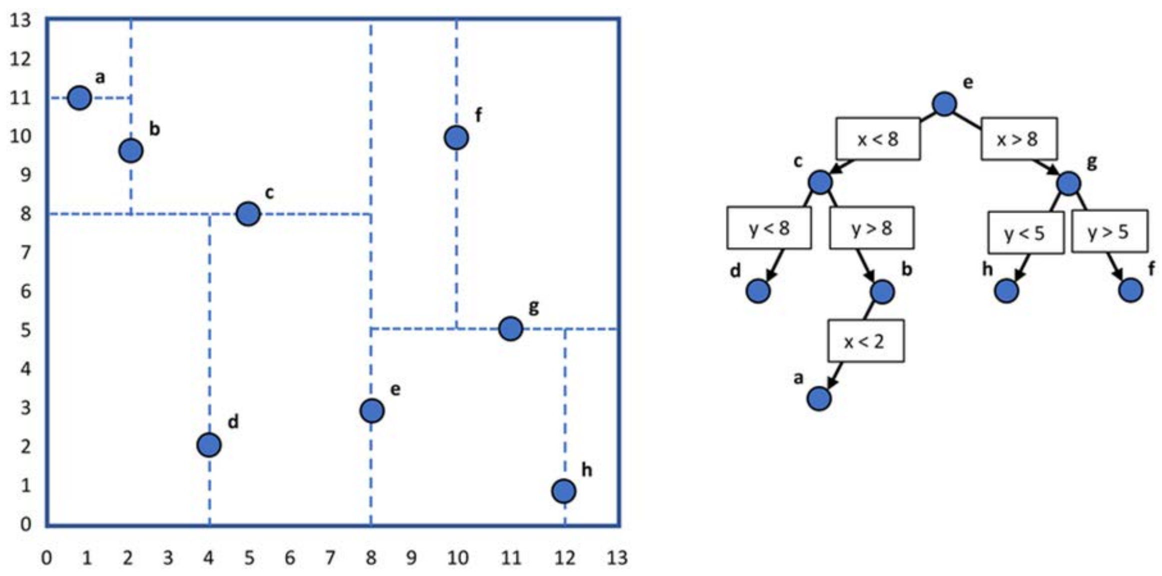


Figure 7. Two-Dimensional KD Tree Example. Source: [24].

The construction of a simple two-dimensional KD tree is depicted in Figure 7. First, the root node is established based on the median value of the first coordinate. Points with smaller first coordinates (horizontal axis) are sorted to the left, larger to the right, creating two child nodes. Then each of those child nodes is partitioned along the second dimension (vertical axis) according to the median value of the second coordinate in each child. This example arrives at one remaining point, the leaf, after two partitions and thus completes the KD tree. The KD tree is a convenient data structure for both range and nearest neighbor searches. Range searches are useful in segmenting LIDAR point clouds. Nearest neighbor KD tree searches are useful in quickly finding nearest point correspondences for ICP.

2. Normal Distribution Transform Method

The Normal Distributions Transform (NDT) Method was initially proposed in [25] as a two-dimensional point registration method. In [26], it was extended to three-dimensional point clouds. Both 2D and 3D NDT registration techniques are used in this work and briefly summarized in this section.

The premise of NDT is relatively simple. Instead of trying to find correspondences between individual closest points, the scan is divided into equal-sized cells, similar to an occupancy map. If a cell contains at least three points, it is assigned a normal distribution. The algorithm then matches consecutive scans by comparing the normal distributions assigned to each cell.

In two-dimensional NDT, after the map is subdivided into cells, the points in each cell are collected into a set. The mean of that set is calculated as is the covariance matrix. The probability of measuring a sample at a specific point contained in the cell is then modeled as a normal distribution.

Additional details on discretization and dealing with nearly singular covariance matrices are included in [25]. In two-dimensional NDT registration, three parameters must be estimated for a transform, they are X and Y translations, and a rotation. The NDT algorithm proceeds as follows [25]:

1. Given two sequential scans in two-dimensional space.
2. Divide the scans into cells of equal size.
3. Assign a normal distribution to each cell of the first scan.
4. Input an estimate for the transformation parameters, this can be done using other sensor data or as zero.

5. In the second scan, map each point into the coordinate frame of the first scan using the parameters estimate.

6. Calculate the normal distributions of the mapped points.

7. Calculate a score for the transformation parameters by evaluating the normal distribution at each mapped point and summing the results.

8. Calculate a new parameter estimate using Newton's algorithm.

9. Iterate until convergence is found.

Three-dimensional NDT is similar to 2D; however, the space is partitioned into voxels instead of cells [26]. Additionally, the transformation consists of six parameters, three translational parameters and three rotational parameters. Discrete 3D points are typically stored in an octree. An octree is a data structure similar to a KD tree, however each node has eight children instead of two, each node represents a partition of the 3D space and it is built recursively similar to KD trees.

NDT is a fast algorithm but slower than ICP. It is, however, more robust to outliers and can be useful for moving object filtering [27].

3. Coherent Point Drift Method

Coherent Point Drift Method (CPD) is a method that uses the Gaussian Mixture Model (GMM) centroids to compute a transform. CPD is a global and robust registration method that can produce a rigid, affine, or non-rigid transformation. A detailed explanation can be found in [28].

The benefit of CPD is that it does not rely on an initial transform estimate, such as ICP or NDT [27, 28]. In general, it is a more accurate registration method than ICP [28]; however, it is considerably more computationally expensive and is the slowest of the three algorithms discussed here [27]. This algorithm is included in the MATLAB Computer Vision Toolbox and was considered for its accuracy. It performed far too slow for an autonomous robot to map in real time and thus was not included in development beyond the initial comparison.

4. Comparison and Selection of a Registration Method

The Computer Vision Toolbox includes functions for point cloud registration using the ICP (3D), NDT (2D & 3D), and CPD (3D) algorithms. To compare the performance of the algorithms several scripts were written. These scripts registered and built LIDAR point cloud maps. The registration comparison script used point cloud data from the popular KITTI Vision Benchmark Suite [29]. This benchmark is used in nearly all of the referenced works for comparison of LIDAR and visual odometry methods. For comparison here, the first 245 point cloud frames from the 00 segment were used, these frames covered about 180 m of outdoor street scene including two roughly 90° turns. The benchmark suite provides numerous datasets. For comparison here, the ground truth data and LIDAR point clouds from the Velodyne HDL-64E Laser Scanner were used.

Output of the comparison script can be seen in Table 1, and mean values are taken from 10 iterations of each registration script running on a desktop computer with a 3.6 GHz AMD Ryzen 5 CPU and 16 GB RAM. The ICP algorithm was by far the fastest method of point cloud registration, more than five times faster than NDT, and seven times faster than CPD.

Table 1. Comparison of Point Cloud Registration Methods

	ICP	NDT	CPD
Mean Rate (Hz)	4.1792	0.8154	0.5650

An example 3D map generated from the ICP script is seen in Figure 8. All three maps from the three methods considered were relatively similar. In all three methods, the algorithm had difficulty detecting the gradual incline of the terrain, and thus the ground truth track diverges up from the estimated track. Use of feature point extraction or loop closure methods as in [8],[9], and others could be used to correct this.

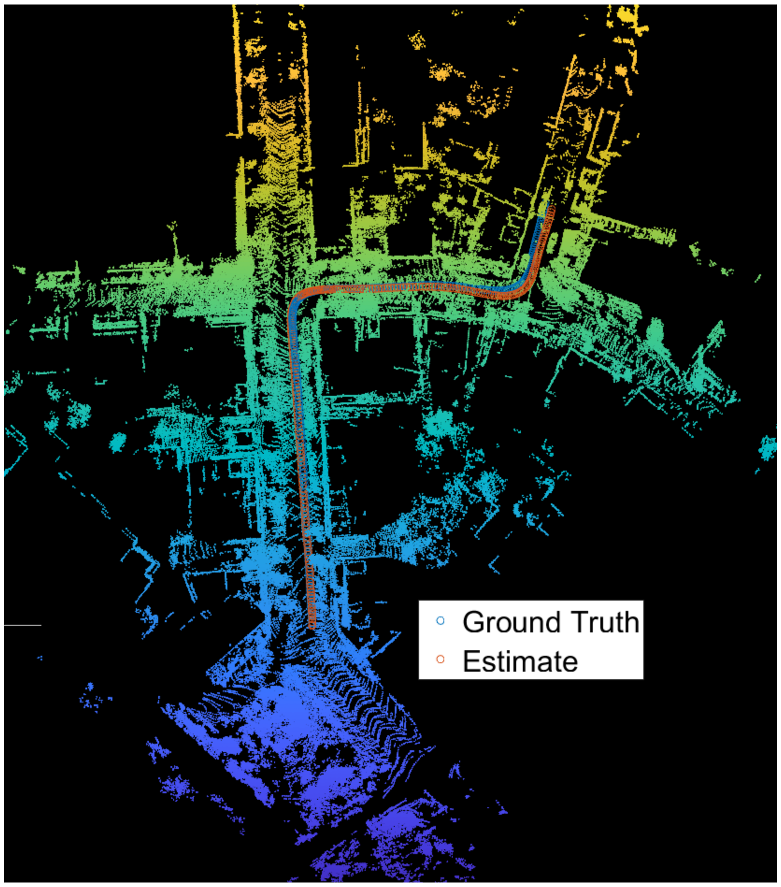


Figure 8. Three-Dimensional Map of KITTI Benchmark Suite Data Generated using ICP.

In the case of both the KITTI Benchmark Suite and the robot constructed for this work, the 3D LIDAR is mounted on top of the platform roughly parallel to the ground plane. Therefore, estimating rotation between LIDAR scans is primarily a rotation about the Z-Axis (Yaw).

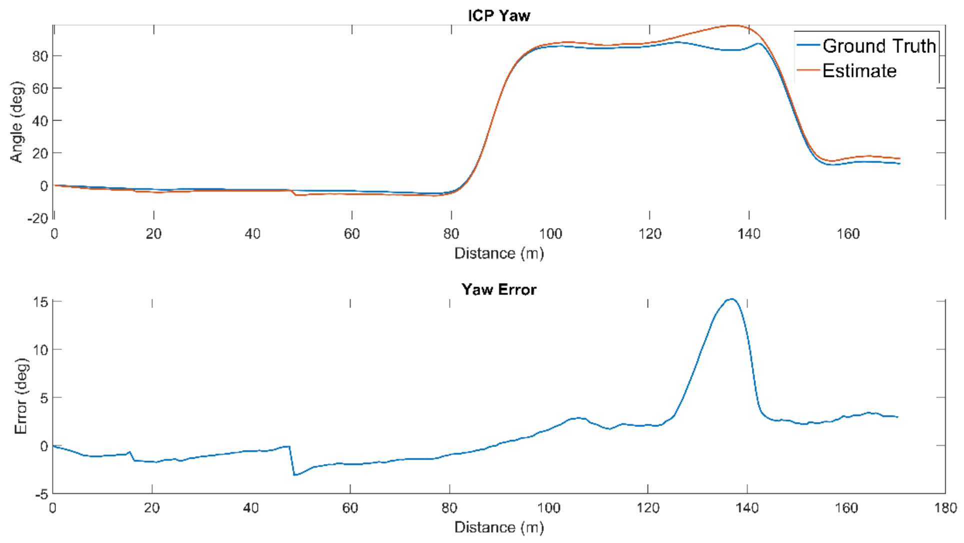


Figure 9. ICP Comparison of Yaw from KITTI Dataset

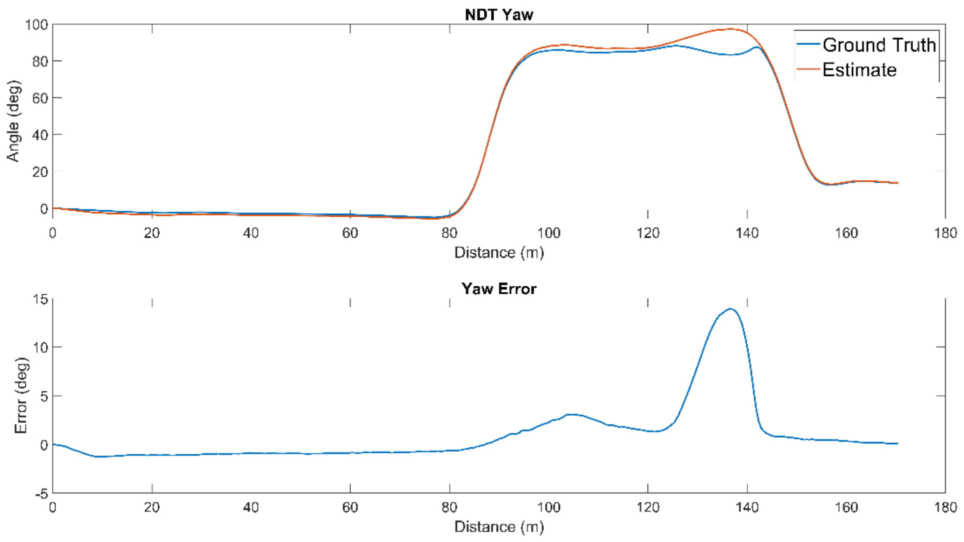


Figure 10. NDT Comparison of Yaw from KITTI Dataset

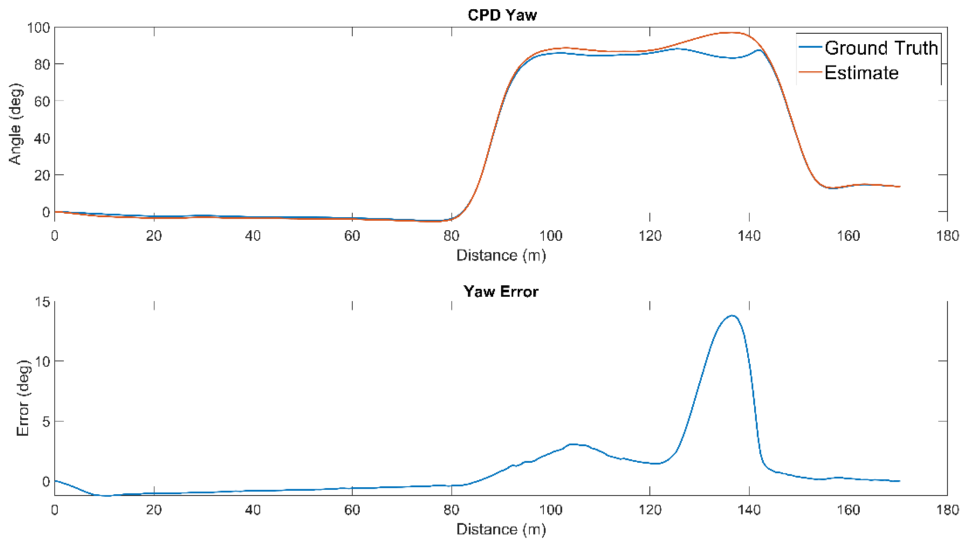


Figure 11. CPD Comparison of Yaw from KITTI Dataset

Yaw and error in yaw are depicted in Figures 9,10, and 11 for the ICP, NDT, and CPD algorithms, respectively. These figures show how the estimated orientation tracks the ground truth orientation as well as the error between the two tracks. The NDT and CPD algorithms have very similar errors between the estimated and ground truth yaw. Maximum errors for both occur during the second turn, 13.9° and 13.8° errors for NDT and CPD, respectively. The ICP algorithm does not track the ground truth data as closely and has a slightly higher error of 15.3° during the second turn.

In the context of this work, speed is the most important parameter under evaluation. Both NDT and CPD performed slightly better in estimating orientation; however, they did so at a much slower rate. Without drift correction, all three algorithms drifted significantly over the course of 180 m. For the application under investigation in this work, the KITTI maps generated (roughly 200 m × 300 m) are much larger than the type of targets of interest. On a smaller scale, the estimated position drift from ground truth is less significant and could be ignored in the interest of saving computational processes that would slow down the LLAM algorithm.

The iterative closest point method was selected for 3D point cloud registration for this work. However, NDT proved both accurate, and extremely fast in 2D. With that knowledge, an ICP algorithm assisted by an initial transform from 2D NDT was developed.

C. POINT CLOUD MERGING AND MAP BUILDING

Creating a map from point cloud frames is essentially a process of applying a transformation to each successive scan to align the point clouds, then combining the point clouds into a map. In this work, transformed point clouds are denoted

$$ {}^{w}P_k = {}^{R}P_k \, {}^{w}_{R}T_k , \tag{6}$$

where ${}^{w}P_k$ is the XYZ point cloud of the k^{th} frame in the {WORLD} reference. ${}^{R}P_k$ is the k^{th} point cloud in the {ROBOT} reference. The k^{th} point cloud in the {ROBOT} frame is transformed into the {WORLD} frame by the absolute transformation ${}^{w}_{R}T_k$.

To simplify calculations, the translational and rotational components are combined in a MATLAB object resembling a homogenous transformation matrix

$$ T = \begin{bmatrix} & & & 0 \\ & R^T & & 0 \\ & & & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}, \tag{7}$$

where R is the 3×3 rotation matrix as previously described and the values of t are the translations along their respective axes. In this form, the accumulated absolute transformation for each successive point cloud can be denoted

$$ {}^{w}_{R}T_k = {}^{R_{k-1}}_{R_k}T_k \, {}^{w}_{R}T_{k-1} , \tag{8}$$

where ${}^{w}_{R}T_k$ is the absolute transformation that maps the k^{th} point cloud from the {ROBOT} frame into the {WORLD} frame. ${}^{R_{k-1}}_{R_k}T_k$ is the transform that maps the current point cloud in the {ROBOT} frame to the previous point cloud in the {ROBOT} frame. This value is the output of the ICP algorithm and also represents the pose of the robot relative to the last calculated pose. ${}^{W}_{R}T_{k-1}$ is the absolute transform that mapped the previously scanned point cloud from the {ROBOT} frame into the {WORLD} frame.

Using this method, as a scan is acquired, it is registered to the previous scan yielding a relative transform, that relative transform is multiplied by the absolute transform from all previous scans, then the point cloud is rotated and translated by the updated absolute transform to be added to the map.

Map building can be accomplished in many ways. In [11], for example, the map is built only out of edge and planar feature points saved after extraction from the raw point cloud. This has the benefit of increasing the speed of that algorithm to 20 Hz, but reduces the density of the point cloud map [11]. In [8], Zhang and Singh employ a method in which the registration algorithm is running at a higher frequency than the mapping algorithm, then the map is downsized using a voxel grid filter to evenly distribute points within the map.

In this thesis, a simple but effective approach is taken. First, a plane is fit to points in a point cloud where the normal vector of the plane is a unit vector in the +Z direction, the indices of those points are segmented and removed from the point cloud and the remaining outliers are stored in a temporary point cloud. This removes the ground plane from the scanned point cloud and assists both the accuracy and speed of the ICP algorithm. The temporary point cloud is then registered to the previous temporary point cloud to yield the relative transform, which is in turn multiplied by the accumulated absolute transformation. The absolute transformation for that scan, and the original point cloud (including the ground plane) are then stored in an object to be compiled into a map later. Each scan is down sampled independently using a grid average method which divides the point cloud into voxels and merges points within the box into a single point, averaging the point normals.

When the robot has run its course, the object storing the point clouds and their associated transforms is then compiled into a map based on a specified voxel grid size to ensure even point distribution. Noise is then removed from the final map using a nearest-neighbors approach as in [30].

D. LOOP-CLOSURE AND DRIFT CORRECTION

Loop-closure is the recognition of re-observed places in SLAM [9]. In LIDAR based SLAM, small errors in the point cloud transformations gradually accumulate and cause the robot pose estimate to drift from the ground truth pose. This is often corrected using loop-closure and back-end optimization, which recalculates the trajectory when a loop-closure is detected and corrects drift [9].

Various methods have been developed for loop closure. In this work, 2D NDT loop closure was attempted using the Hokuyo sensor. MATLAB includes a SLAM package that can support LIDAR data. This package was used to compute 2D pose estimates from the Hokuyo data, probabilistic occupancy maps, and loop closures. Additionally, when loop closures are detected, pose optimization can be performed to recalculate pose estimates and rebuild the map correcting for drift. In early experimentation, these functions were integrated into the robot LLAM algorithm. However, on further evaluation, they were removed for several reasons.

First, loop-closure detection was too computationally expensive to run in real time on the robot. In order to search for loop-closures, the robot had to stop at intermediate waypoints and compute whether a loop-closure was present. Second, after significant experimentation, a loop closure was never detected. This was not a problem with the algorithm, but rather in the employment of the robot. The robot developed here, is designed from a tactical perspective. The purpose of building a map is not to examine every unknown space or create a highly detailed and accurate map. The purpose here is to collect a suitable map as quickly and efficiently as possible. Therefore, the robot never revisited previous locations, it was not programmed to do that, and therefore did not detect loop-closures. Similarly, loop-closure in 3D as in [9] was not examined as the robot was unlikely to return to a previously visited location. Furthermore, the limited FoV of the Pico monstar make detecting previously visited locations in 3D extremely difficult as the robot must be in both the same position and also have a similar orientation.

Finally, within the bounds of the ECE Control Systems Laboratory and adjacent spaces, the robot did not drift significantly enough to be identified without additional sensors estimating the pose. That is, the map was accurate enough that a human operator could not distinguish it visibly from the scene it was representing. Also, the error in pose was not significant enough to cause the robot to miss waypoints. The obstacle avoidance algorithm ran every loop iteration and computed steering parameters off every local frame. Thus, drift from the ground truth pose would not contribute to a collision of the robot and any obstacle in the environment. As the tactical premise of this robot involved revisiting locations as infrequently as possible, loop-closure based drift correction was determined to be unnecessary. Development of a loop-closure detection algorithm that is triggered only when the robot must return to a previously visited location, such as entering a room with only one exit, is left to future work.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ROBOT CONTROL SCHEME

A. OVERVIEW

As discussed in Chapter II, two robots were constructed. Both robots used the same P3-DX base robot, Pico monstar 3D LIDAR, and Hokuyo 2D LIDAR. The first robot (Robot 1) carried a Windows OS Dell laptop attached to the top of the robot as seen in Figure 12. On the second robot (Robot 2), the Dell laptop was replaced with the SlimPRO computer communicating wirelessly to the Dell desktop. Robot 1 was constructed while attempting to solve a Linux compatibility issue with the Pico monstar MATLAB wrapper. This compatibility issue initially prevented it from interfacing with the Linux based SlimPRO computer.

The design of Robot 1 did not satisfy the tactical military intent of this work. It could only partially answer the central question of whether a 3D mapping robot can use only LIDAR in real time. Since no deployable, tactical sized robot would carry a full-size laptop as a processing unit, Robot 1 would not be satisfactory. However, some valuable insights were gained from the development of this robot and are worth examining.

The laptop carried on Robot 1was far more powerful than the SlimPRO and allowed for the testing of several SLAM algorithms that were not included on Robot 2. First, the more powerful computer allowed for six DoF point cloud registration to run at nearly 10 Hz when assisted by an initial 2D NDT registration from the Hokuyo. Secondly, on-board occupancy map building, Hybrid A*Search route planning, loop-closure detection, and frontier search were implemented. Even with the more powerful computer, the occupancy map building and loop-closure detection required the robot to stop at intermediate waypoints for times ranging between 10–60 seconds in order to run through the algorithms.

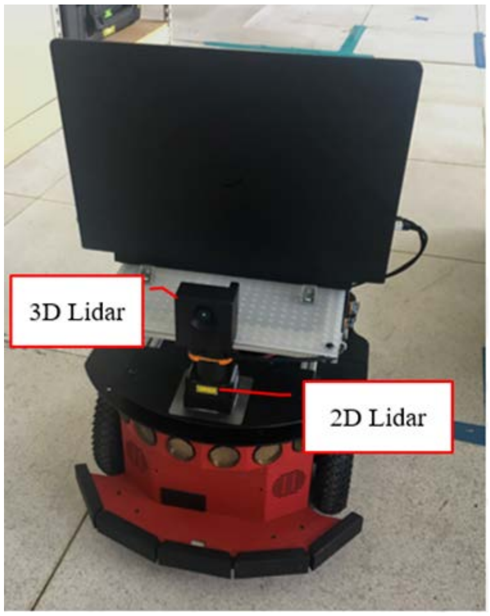
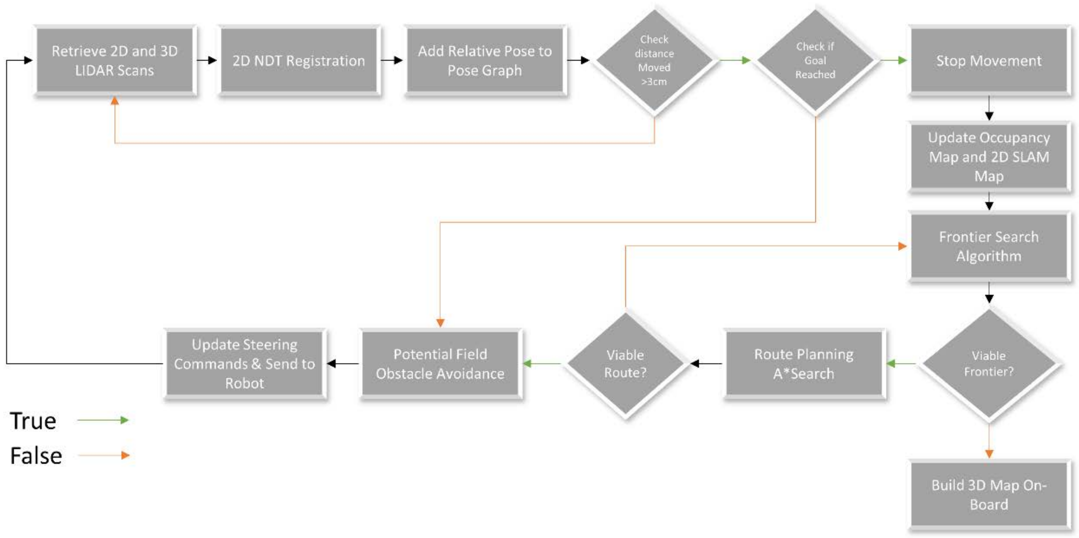


Figure 12. Robot 1

Several important things were learned from experimentation with Robot 1. First, loop-closure is not necessary in the type of small interior spaces that are targeted in this work. Second, for a ground mobile robot, 3D registration was significantly slower than 2D registration but returned results that were minimally more accurate. That is, for the tactical operator, the distortion in the map between 2D registration and the more accurate 3D registration was not distinguishable. Furthermore, the 2D registration was perfectly satisfactory for the robot to navigate successfully and avoid obstacles.

Finally, as previously mentioned, an architecture in which all data is processed on-board the robot is ill suited for tactical situations. If the building is occupied by hostile actors, the robot is unlikely to emerge. In this case, the map would be lost. If the robot transmits the relevant map-making data and it is disabled prematurely, at least a partial map can be constructed on a remote computer. The LLAM algorithm developed for Robot 1 is depicted in Figure 13.



Retrieve 2D and 3D LIDAR Scans
2D NDT Registration
Add Relative Pose to Pose Graph
Check distance Moved >3cm
Check if Goal Reached
Stop Movement
Update Occupancy Map and 2D SLAM Map
Frontier Search Algorithm
Viable Frontier?
Route Planning A*Search
Viable Route?
Potential Field Obstacle Avoidance
Update Steering Commands & Send to Robot
Build 3D Map On-Board
True
False

Figure 13. Robot 1 LIDAR Localization and Mapping Algorithm

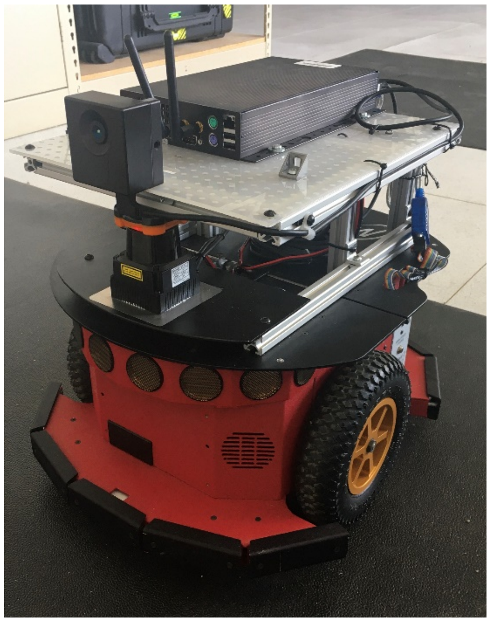


Figure 14. Robot 2

The hardware of Robot 2 is shown in Figure 14, while the design architecture is illustrated in Figures 15 and 16. The onboard SlimPRO retrieves 2D and 3D LIDAR data from the sensors. It uses the 2D data for pose estimation and navigation. The 2D and 3D

LIDAR data is combined for obstacle avoidance, and the 3D point cloud data along with the 2D pose estimate are transmitted to the remote computer for map building.

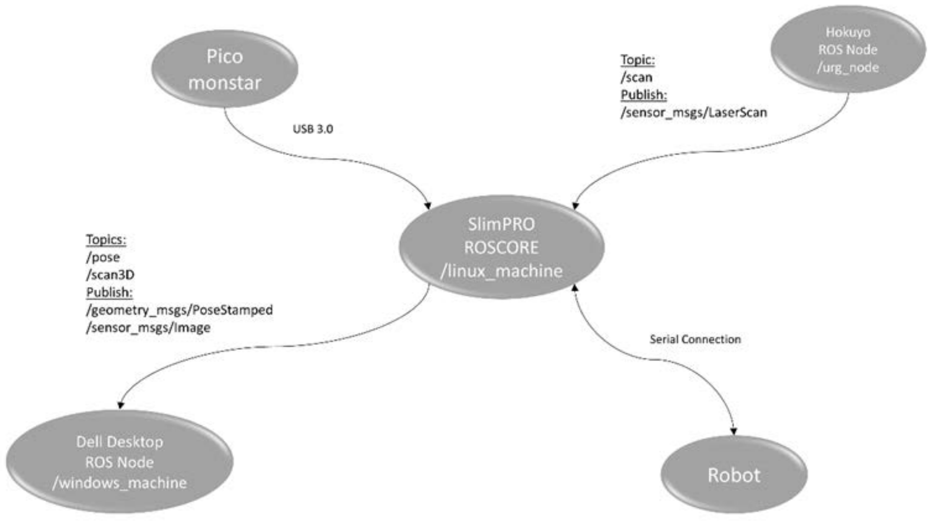

Pico monstar
USB 3.0
Hokuyo ROS Node /urg_node
Topic: /scan
Publish: /sensor_msgs/LaserScan
SlimPRO ROSCORE /linux_machine
Topics: /pose /scan3D
Publish: /geometry_msgs/PoseStamped /sensor_msgs/Image
Serial Connection
Dell Desktop ROS Node /windows_machine
Robot

Figure 15. Robot 2 ROS Architecture

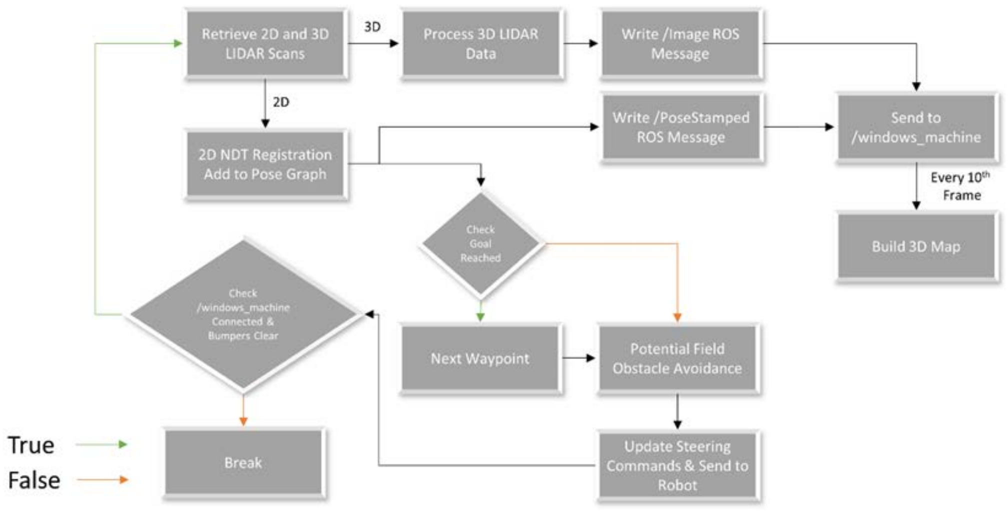

Retrieve 2D and 3D LIDAR Scans
3D
Process 3D LIDAR Data
Write /Image ROS Message
2D
2D NDT Registration Add to Pose Graph
Write /PoseStamped ROS Message
Send to /windows_machine
Every 10th Frame
Build 3D Map
Check Goal Reached
Check /windows_machine Connected & Bumpers Clear
Next Waypoint
Potential Field Obstacle Avoidance
Update Steering Commands & Send to Robot
Break
True
False

Figure 16. Robot 2 LIDAR Localization and Mapping Algorithm

B. OBSTACLE AVOIDANCE

In this work, an artificial potential field model for obstacle avoidance was used. The algorithm is similar to those used in [3],[4], and [5], but in this case, was adapted to three dimensions. The model assigns an attractive potential between the robot and the goal, and it computes a repulsive potential to obstacles that it senses through LIDAR returns. It then calculates the potential field as the sum of the attractive and repulsive potentials. The negative gradient of the potential field yields a force vector that draws the robot closer to the goal while simultaneously avoiding obstacles along the path.

1. Attractive Force

Implementation of the attractive force matches that of [33] adapted to the Hokuyo LIDAR system. The attractive potential U_{att} is defined as

$$U_{att} = \begin{cases} \frac{1}{2}\xi \left\| q - q_{goal} \right\|^2, & \text{if } \left\| q - q_{goal} \right\| \le \rho \\ \xi\rho \left\| q - q_{goal} \right\|, & \text{if } \left\| q - q_{goal} \right\| > \rho \end{cases}, \tag{9}$$

where q represents the XY coordinates of the current pose, and q_{goal} represents the XY coordinates of the goal pose, both in the {WORLD} frame. The terms ξ and ρ are constants, ξ being a constant coefficient and ρ being a constant distance. This creates an attractive potential that is conic shaped when the robot is at least ρ distance from the goal, and parabolic in shape when it is closer [33].

The attractive force ${}^{w}F_{att}$ is given by the negative gradient of the attractive potential as [33]

$${}^{w}F_{att} = -\frac{\partial U_{att}}{\partial q} = \begin{cases} -\xi(q - q_{goal}), & \text{if } \left\| q - q_{goal} \right\| \le \rho \\ -\xi\rho \dfrac{(q - q_{goal})}{\left\| q - q_{goal} \right\|}, & \text{if } \left\| q - q_{goal} \right\| > \rho \end{cases}. \tag{10}$$

2. Repulsive Forces

The repulsive potential is generated by LIDAR returns from obstacles within the LIDAR field of view. Each LIDAR return represents a repulsive potential of the form [33]

$$U_{rep,i} = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d_i} - \frac{1}{d_c}\right)^2, & \text{if } d_i \le d_c \\ 0, & \text{if } d_i > d_c \end{cases}, \tag{11}$$

where d_i is the range measurement taken from an individual LIDAR return, d_c is a constant cutoff distance, and η is a constant coefficient.

The repulsive force from a single LIDAR return is then given in the {ROBOT} frame by [33]

$${}^{R}F_{rep,i} = \begin{cases} -\eta\left(\frac{1}{d_i} - \frac{1}{d_c}\right)\frac{n_i}{d_i}, & \text{if } d_i \le d_c \\ 0, & \text{if } d_i > d_c \end{cases}, \tag{12}$$

where n_i is a unit vector in the direction (angle γ) of the laser return referenced to the {ROBOT} frame. In this form, the repulsive force is inversely proportional to the distance from the obstacle when it is inside the cutoff distance and zero if it is beyond the cutoff distance. The unit vector n_i is defined as

$$n_i = \begin{bmatrix} \cos\gamma_i \\ \sin\gamma_i \end{bmatrix}. \tag{13}$$

The Hokuyo LIDAR data is initially referenced with respect to the LIDAR {HOKUYO} frame in cartesian coordinates. It is translated to the {ROBOT} frame simply by subtracting a translation t_{hx}, the distance from the LIDAR center to the center of the robot. Thus, an individual point ${}^{H}p_i$ is mapped from the {HOKUYO} frame to the {ROBOT} frame as

$$ {}^{R}p_i = [{}^{R}x_i, {}^{R}y_i] = [{}^{H}x_i, {}^{H}y_i] + [t_{hx}, 0]. \tag{14} $$

The values of d_i and γ_i from Equations (12) and (13) can then be calculated as

$$ d_i = \left\| {}^{R}p_i \right\| \quad , \tag{15} $$

and

$$ \gamma_i = \arctan\left(\frac{{}^{R}y_i}{{}^{R}x_i} \right). \tag{16} $$

The total repulsive force is simply the sum of the repulsive forces calculated from the individual LIDAR returns [33]

$$ {}^{R}F_{rep} = \sum_{i=1}^{N} {}^{R}F_{rep,i} \ . \tag{17} $$

In this case, N is the number of LIDAR returns within the cutoff distance and varies greatly based on the environment. The Hokuyo LIDAR typically returns points for nearly all of the laser shots (1080). However, to shorten the iterative loop, values beyond the cutoff distance are simply omitted rather than summing zero vectors, as in Equation (12).

Finally, the total force in the {ROBOT} frame ${}^{R}F_{total}$ is calculated by adding the total repulsive force and the attractive force [33]

$$ {}^{R}F_{total} = [{}^{R}f_{x,total}, {}^{R}f_{y,total}] = {}^{R}F_{rep} + {}^{R}_{w}T(\theta)\,{}^{w}F_{att} \ . \tag{18} $$

The attractive force, which was calculated in the {WORLD} frame, must be transformed to the {ROBOT} frame using the 2D rotation matrix [33]

$$ {}^{R}_{w}T(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{19} $$

where θ is the angle between the X-Axis of the {ROBOT} frame and the X-Axis of the {WORLD} frame, which is simply the third parameter of the 2D pose of the robot determined from NDT scan matching.

The forward velocity and rotational velocity of the robot are then calculated as

$$V_{fwd} = K_f \, {}^{R}f_{x,total} , \tag{20}$$

and

$$V_{rot} = K_r \arctan\left(\frac{{}^{R}f_{y,total}}{{}^{R}f_{x,total}} \right). \tag{21}$$

where K_f and K_r are constant gains associated with the forward and rotational velocities and tuned during experimentation. The rotational and forward velocity commands are then sent from the on board computer to the robot by serial connection using the *p3_setRotVel()* and *p3_setTransVel()* functions included in Appendix C.

3. Adaptation to Three Dimensions

In this work, the combination of 2D and 3D LIDAR allowed for the robot to sense obstacles in three dimensions. Previous thesis work done by Miyakawa in [4] used a second 2D Hokuyo LIDAR to sense low profile obstacles in front of the robot. That LIDAR had a fixed angle at which it was declined, and thus could only see a fixed distance ahead of the robot. In this case, the 3D LIDAR allowed the robot to sense obstacles five to six meters ahead of it and begin adjusting the trajectory via the artificial potential field algorithm to avoid the obstacle smoothly.

Additionally, the model presented in [4] could only sense low profile obstacles on the ground, such as stairs or other obstructions below the horizontal plane of the second Hokuyo LIDAR, which could lead to the robot planning a route underneath an object without sufficient clearance for the sensors and other hardware mounted on top of the robot. In contrast, the Pico monstar LIDAR can sense obstacles from ground level to the ceiling of an interior space at comparatively high resolution to the Hokuyo. This allowed for the potential field algorithm to be assisted by repulsive forces that did not lie in the plane of the Hokuyo LIDAR.

For this work, the 3D point cloud returned from the Pico monstar was sent to the potential field algorithm. It was then cropped to only include points that lay in the vertical (Z) dimension between 0.07 m and 0.45 m. The first constraint being the clearance of the robot, the second being the top of the Pico monstar sensor.

Since the ground mobile robot can only move in three DoF, considering repulsive forces in three dimensions would not be useful as the vertical component of those repulsive forces would not contribute to the output steering commands. Instead, the X and Y coordinates of any 3D LIDAR return within the previously described height bounds were added to the array of returns from the Hokuyo LIDAR. This essentially compressed the third (Z) dimension of the 3D LIDAR returns down into the XY plane of the {HOKUYO} frame.

When compressing the 3D LIDAR returns into an array of two-dimensional coordinates, it is possible to have duplicate 2D points. Summing of the repulsive vectors based off these duplicate points will create a total repulsive force that has a much larger magnitude than it should to accurately avoid the obstacle. Essentially, the potential field algorithm calculates a larger obstacle than is actually present. In order to correct this, 3D LIDAR data was first divided into bins corresponding to the angular resolution of the Hokuyo LIDAR. From those bins, only one point from the 3D LIDAR return was selected to be added to the 2D LIDAR data. This simple yet effective method removed the need for comparative loops to determine if two nearby points were different enough to significantly affect the repulsive potentials. Additionally, it weighted the 3D point returns equally with the 2D point returns removing an additional tuning requirement for the algorithm.

C. NAVIGATION

Several methods of navigation were used throughout development of the robots for this work. First, pre-determined waypoints were used both with a Hybrid A* Search algorithm and with a pure pursuit algorithm. Second, a frontier search algorithm was developed to find unexplored areas of the map without any pre-determined waypoints.

1. Pure Pursuit

During pure pursuit navigation, the robot has a known goal or list of goals and attempts to take the most direct route to the goal while avoiding intermediate obstacles. This method has disadvantages when combined with potential field-based navigation. While attempting to reach the goal, the robot can encounter a local minimum in the potential field and become trapped, as described in [3]. If the robot becomes trapped in a local minimum, an additional set of control logic is required to help it escape and continue moving towards the goal. Previous thesis work, such as [4] and [5], used wall following and terrain following modes of escaping local minima. For this work, the primary purpose was to determine if the robot could conduct 3D SLAM in real time. During experimentation, the robot was positioned within the laboratory space such that it had obstacles, but not the kind that would create local minima in the potential field. For example, obstacles that are concave U-shaped from the perspective of the robot, or interior corners between the robot and the goal can create local minima. Instead, while using pure pursuit navigation, obstacles were limited to objects such as road cones, paint buckets, chairs, outside corners, etc. Robot 2 only implemented pure pursuit navigation during experimentation.

2. Hybrid A* Search

The MATLAB Navigation Toolbox includes various path planning algorithms. Included in the toolbox is a Hybrid A* Search algorithm based on the work in [34]. This algorithm was implemented on Robot 1 both with pre-determined waypoints and a frontier search algorithm. The Hybrid A* Search algorithm works by discretizing the known obstacles in the environment into an occupancy map and searching based off motion primitives specific to the robot [34]. An occupancy map divides the known environment into cells and assigns a tag to the cell as either “occupied,” “unoccupied,” or “unknown.” MATLAB has several tools for generating occupancy maps. In this work, the MATLAB SLAM algorithm was used to compute 2D NDT transforms of successive Hokuyo laser scans, then combine them into a map. The world pose of the robot was added to a pose graph, and the ternary occupancy map was then built from the SLAM map.

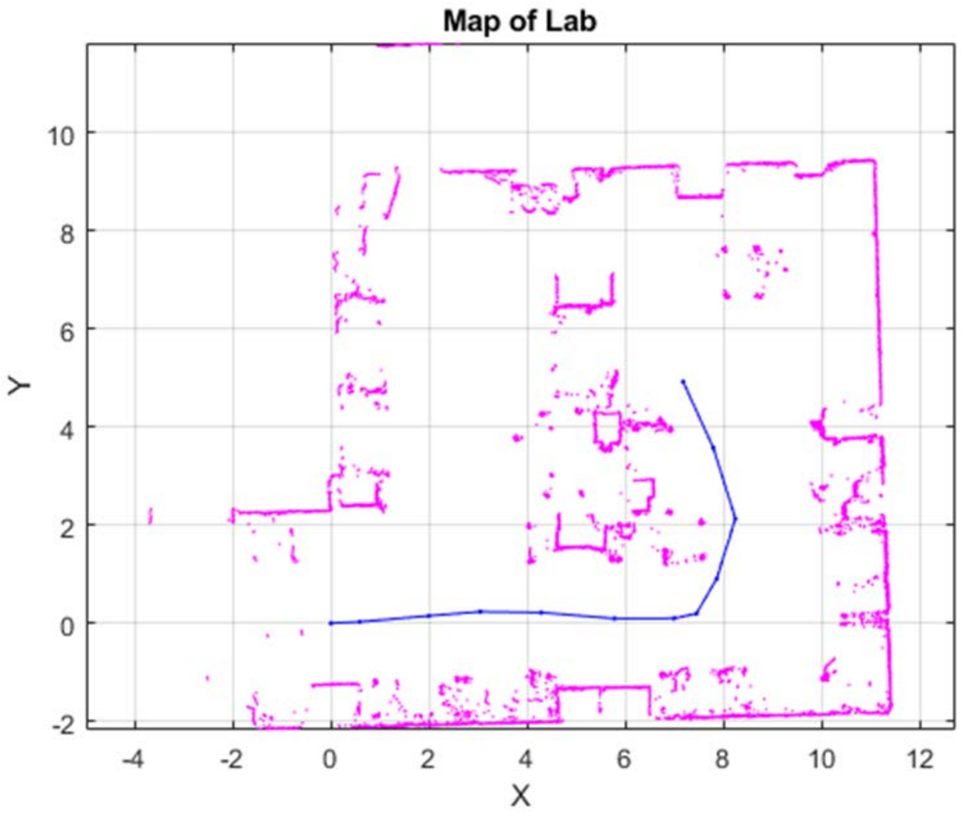


Figure 17. Robot 1 SLAM Map (Spanagel Hall Laboratory 521)

The pink points in Figure 17 are discrete laser returns from the Hokuyo sensor and the blue track is the trajectory of the robot computed from the transformations of successive laser scans. An occupancy map of the lab environment computed from this SLAM map is shown in Figure 18.

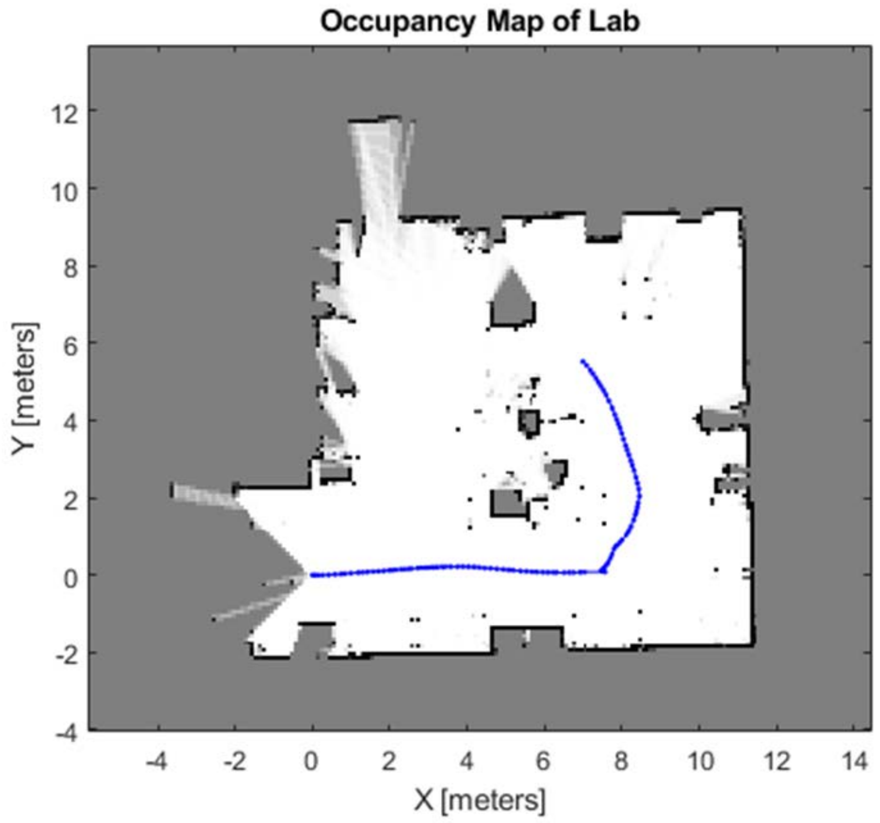


Figure 18. Robot 1 Occupancy Map (Spanagel Hall Laboratory 521)

The LIDAR data from the SLAM map is then divided into cells and labeled accordingly. The white cells in Figure 18 correspond to space that is known to be unoccupied by obstacles. Black cells correspond to known occupied space, and the gray cells represent unknown space that the robot has not yet explored or cannot see.

The Hybrid A* Search algorithm proceeds from this point with the current pose of the robot, the maneuver capabilities of the robot, and the occupancy map as inputs. The algorithm calculates the paths possible to reach adjacent cells terminating in a node, then from those nodes calculates the next set of possible paths. In contrast to the standard A* Search, Hybrid A* Search takes into account nonholonomic constraints of the robot, such as turning radius [34]. An example is shown in Figure 19.

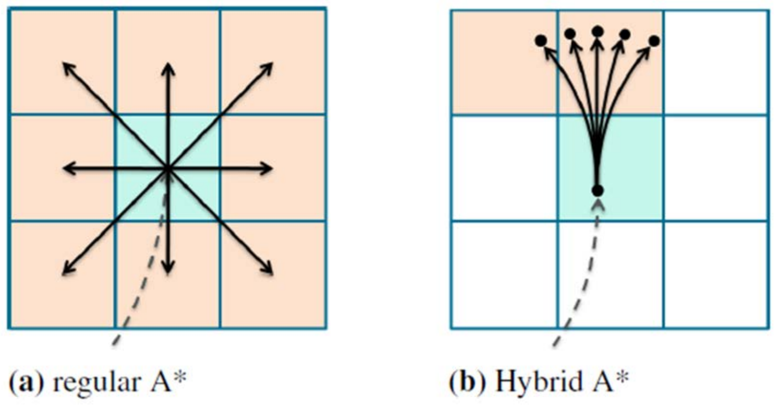


Figure 19. A* Search versus Hybrid A* Search. Source: [34]

The algorithm then uses a cost function to compute the least costly route to the goal. Figure 20 illustrates an example of the Hybrid A* Search algorithm as it calculates all the possible nodes from each previous node and selects the least costly path. Many of the nodes terminate because of the nonholonomic constraints of the robot.

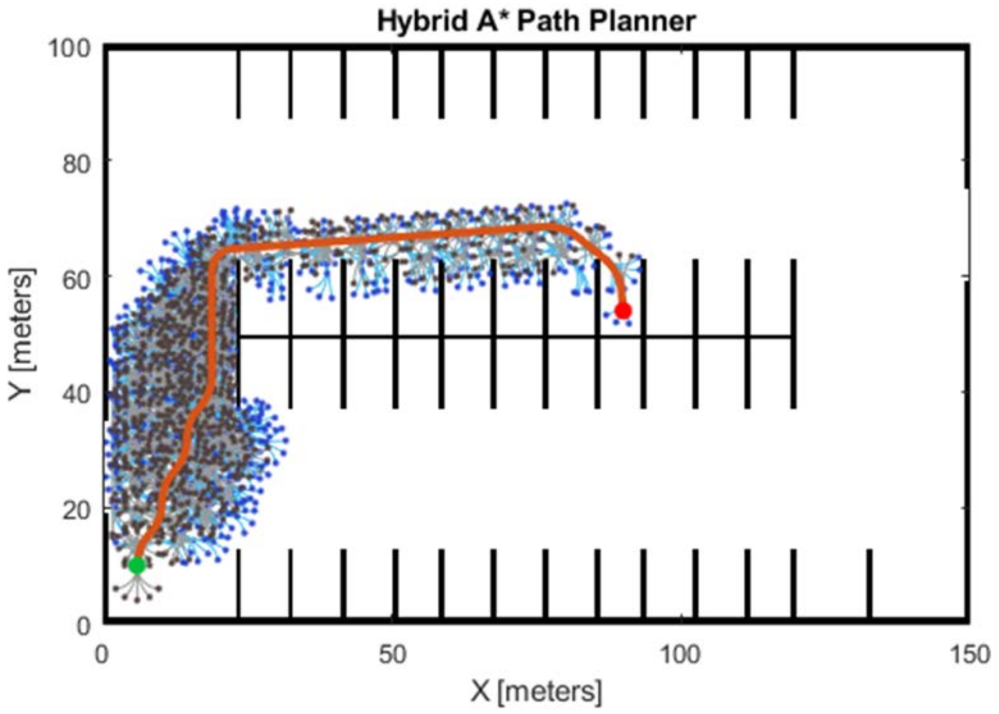


Figure 20. Example Implementation of Hybrid A* Search Algorithm in MATLAB. Source: [35]

In this work, Robot 1 utilized the MATLAB Hybrid A* Search algorithm to find and plan paths. The calculated nodes of the path were set as intermediate waypoints and the robot traveled to them utilizing the potential field algorithm to negotiate any dynamic or previously unidentified obstacles. This implementation was fairly effective; however, it was ultimately not included on Robot 2. The primary issue was the time required to build the SLAM map and occupancy map, as inputs for the Hybrid A* Search algorithm. In most cases, MATLAB does not allow multiple processes to run simultaneously. The sequential nature of the script meant that LIDAR scans were not being received while the necessary path planning was occurring. If the robot remained in motion, scan matching would degenerate and the robot would lose localization, a problem further discussed in Chapter V. In order to prevent this, the robot came to a stop for the necessary amount of time to build the SLAM and occupancy maps and plan an obstacle-free path. Typically, this stop was between 10 and 60 seconds. This method was implemented on Robot 1, which had a full-size laptop attached in place of the SlimPRO. If implemented on Robot 2, the delays would have been more significant. Thus, building the on-board maps and planning the route as simultaneous processes are left to further work, likely requiring migration to a different programming language.

3. Frontier Search

In order to truly explore a new environment, a robot must be able to determine what parts of the immediate environment have not been explored. Once unexplored areas of the map have been identified, one must be selected based on specified criteria for exploration. The robot should then use already existing knowledge of obstacles, as well as the current pose information to plan a path to explore the frontier. There are many ways to achieve this and many criteria by which the robot can select a frontier. MATLAB does not include any frontier search or exploration algorithms, but one was created for this work using image processing techniques.

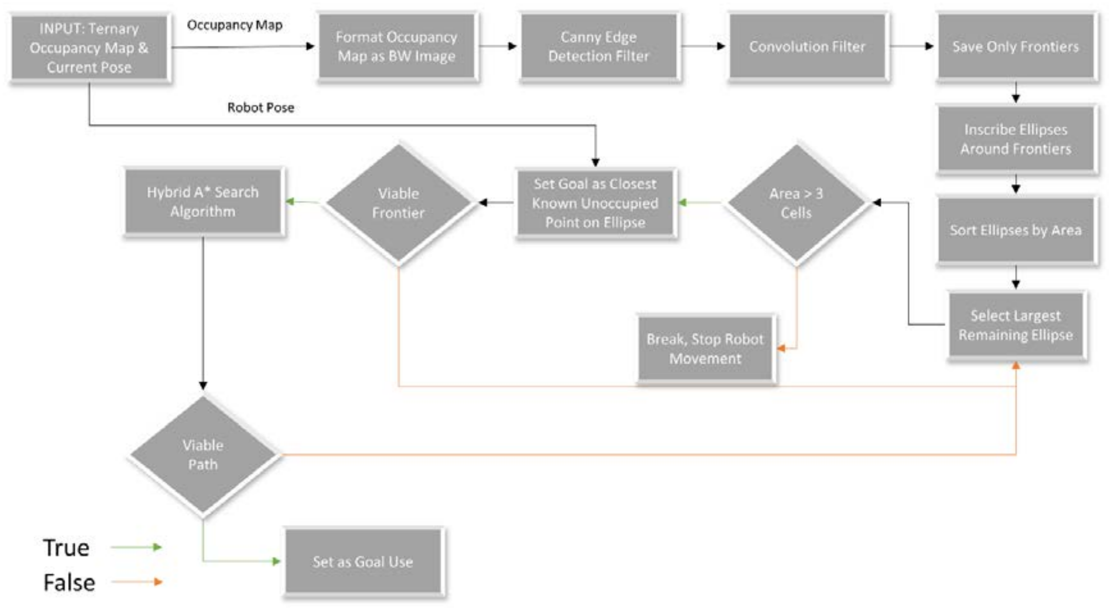

INPUT: Ternary Occupancy Map & Current Pose
Occupancy Map
Format Occupancy Map as BW Image
Canny Edge Detection Filter
Convolution Filter
Save Only Frontiers
Robot Pose
Inscribe Ellipses Around Frontiers
Hybrid A* Search Algorithm
Viable Frontier
Set Goal as Closest Known Unoccupied Point on Ellipse
Area > 3 Cells
Sort Ellipses by Area
Select Largest Remaining Ellipse
Break, Stop Robot Movement
Viable Path
True
False
Set as Goal Use

Figure 21. Frontier Search Algorithm

A flow chart of frontier search algorithm is depicted in Figure 21. The algorithm takes as inputs the current pose of the robot and the ternary occupancy map that was compiled when the robot stopped at the initial goal. It formats the occupancy map, as seen in Figure 22, as a gray scale image and uses a Canny edge detector algorithm to find the pixels in which there is a transition from between white, gray, and black. These edges however indicate a transition between the three colors present in the gray scale image. Frontiers are the transition between known unoccupied space (white) and unknown space (gray) only. The transition from known occupied space (black) to known unoccupied space (white) is of no interest when searching for frontiers. In order to remove those edges, a convolution filter is run to identify all the edge pixels that are a transition from known unoccupied space to unknown space only.

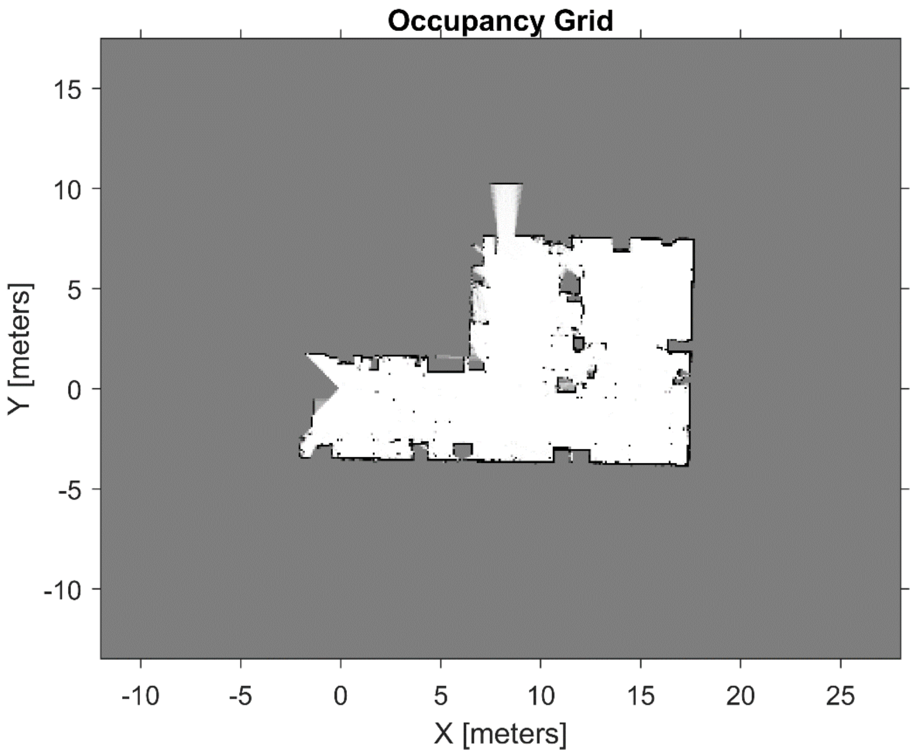


Figure 22. Example Initial Ternary Occupancy Map
(Spanagel Hall Laboratory 521)

The remaining edges are then saved in a binary image as frontier pixels. If multiple frontier pixels are adjacent to one another, those pixels are grouped together as being connected and the group is saved as a frontier. An example of connected frontier pixels which are saved in a binary image is depicted in Figure 23.

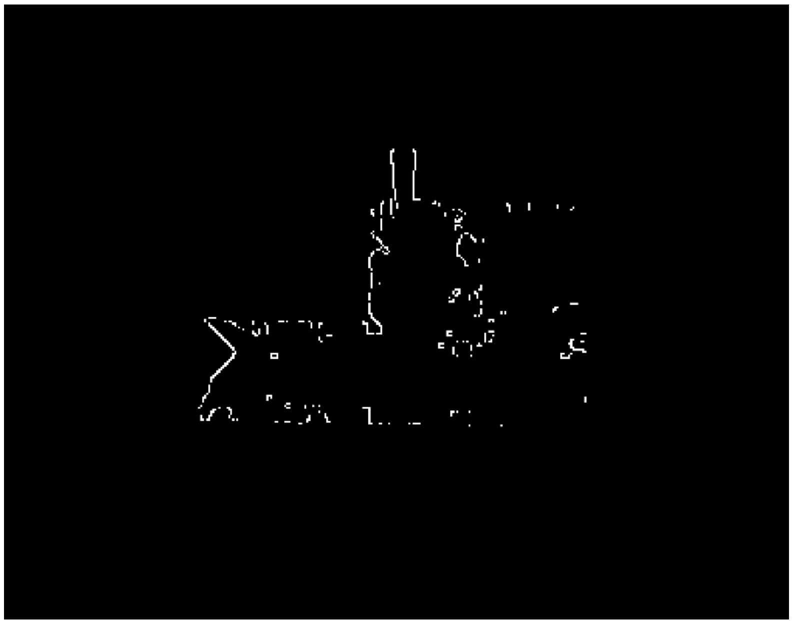


Figure 23. Example of Remaining Frontier Pixels (Spanagel Hall Laboratory 521)

The connected frontiers are then inscribed with ellipses as seen in Figure 24. The parameters of each ellipse are saved in a data structure, which is sorted from the largest area to the smallest area. In order to be a valid state for the Hybrid A* Search algorithm, the centroid of the ellipse cannot be used as a goal because it may lie in unknown space. To find a valid goal, the algorithm selects the points that lie on the ellipse and are known to be cells of unoccupied space. Using these points, it computes the Euclidean norm from the current robot pose to each point. The ellipse point with the minimum distance is selected as a temporary goal. It is mapped from image coordinates back into world coordinates and sent to the Hybrid A* Search algorithm. If a valid path is found, that temporary goal is assigned as the robot goal and the LLAM algorithm continues. If a valid path is not found, the frontier search algorithm selects the next largest ellipse and repeats the process until a valid path is found. If no valid paths are found and the area of the remaining ellipses is less than three pixels (occupancy map cells), the algorithm terminates, and the robot has completed exploration.

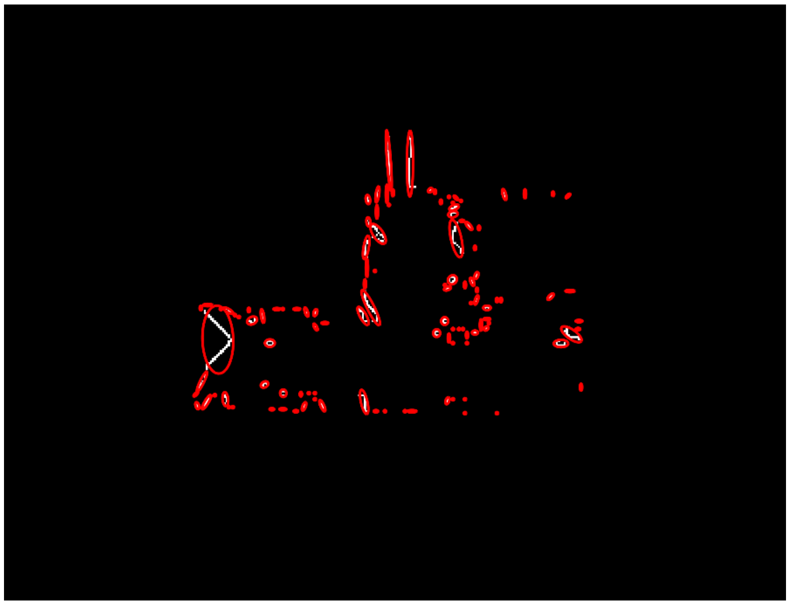


Figure 24. Example of Ellipses Inscribing Connected Frontiers (Spanagel Hall Laboratory 521)

This algorithm was successfully employed on Robot 1 and ran relatively fast after the already discussed time delay required to build the occupancy map. Depending on the amount of unexplored area remaining in the laboratory, it took between 0.3 and 3 seconds to find a viable path. One major disadvantage of this algorithm is that it always seeks the largest frontier first. In some cases that causes the robot to wander around the test space, back tracking already traveled space several times. On the other hand, using the closest unexplored frontier is less effective because it can be very small and not significant in the overall map. Additionally, some frontiers cannot be reached to be explored due to obstacles. As the Hybrid A* Search algorithm does not incrementally re-plan the route while the robot is moving, the robot can become trapped in a point where a previously unrevealed obstacle is preventing forward progress due to potential field obstacle avoidance algorithm, which is running in stride as part of the LLAM algorithm. That is, the robot could not see an obstacle when the route was planned to the frontier, but upon trying to travel that route, it encountered an obstacle. The Hybrid A* algorithm will not re-run until it reaches the goal, but the potential field algorithm will not let it continue on that route and reach the goal. This is another instance of the local minima problem already discussed.

Although the frontier search algorithm, as presented, can run fast enough to be incorporated in real-time LLAM, it relies on the occupancy map as an input. For reasons already discussed, that did not prove feasible. Thus, the frontier search was not included when transitioning to Robot 2. It is included here because this novel approach shows some promise for future development. If a faster occupancy map method, local minima escape logic, or a different obstacle avoidance model can be implemented, it may prove useful for indoor exploration.

D. MAP BUILDING

1. ROS Network and Point Cloud Transmission

When Robot 2 was transitioned to the wireless ROS network, it was confirmed that point cloud messages are too large to be transmitted and collected in real time. If the Robot mapping algorithm is running faster than approximately 1 Hz, point cloud messages can be lost or received out of sync with pose messages. This was expected due to the size of 3D point clouds. However, significant overlap in the 3D point clouds meant that not every point cloud must be received in order to build an accurate map. In this work, every tenth 3D point cloud was transmitted with the corresponding 2D pose estimate.

Additionally, it was discovered that organized 3D point cloud data could be transmitted more efficiently using a three-channel ROS image message *sensor_msgs/Image* instead of the more common *sensor_msgs/PointCloud2* message. The organized point cloud is retrieved from the Pico monstar sensor in a format resembling a single floating-point precision three-channel image. Therefore, the image message could be written directly from the received data without reformatting. Two-dimensional pose estimates were sent as *geometry_msgs/PoseStamped* messages, which include a header that can contain a frame ID number. Two methods of building the map on the remote desktop computer were implemented and are described in subsequent sections.

2. Two-Dimensional Transformation Method

The first method utilized two-dimensional scan data received from the Hokuyo LIDAR. This 2D data was immediately registered using NDT registration to calculate a relative transformation. The absolute transformation accumulated as described previously, then describes the 2D pose of the robot. With the exception of vibration, the roll and pitch of the LIDAR sensors on the robot are constrained. The only significant orientation changes occur about the Z-Axis in the {ROBOT} frame. One method of constructing the 3D map is simply create a 3D transformation using the absolute transformation of the robot in 2D. Given the 2D {WORLD} pose of the robot

$$^{w}p_k = [x_k \quad y_k \quad \theta_k], \tag{22}$$

the 3D transformation to the {WORLD} frame can be computed as

$$^{w}_{R}T_k = \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 & 0 \\ \sin\theta_k & \cos\theta_k & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_k & y_k & z_{PMD} & 1 \end{bmatrix}, \tag{23}$$

where z_{PMD} is the fixed height of the 3D Pico monstar LIDAR sensor. Each successive 3D LIDAR point cloud is then transformed by the corresponding {WORLD} transformation, which was calculated only from the data returned by the 2D Hokuyo sensor. The resulting transformed point clouds can then be merged together to form a map. The final map is down sampled using a voxel grid filter.

This method is beneficial because it is much faster than performing iterative closest point registration in three dimensions. Additionally, it relies on 2D transformations that are already computed as part of the navigation algorithm. On the other hand, it assumes that the pitch and roll of the robot are constrained and is therefore only applicable to ground mobile robots with three DoF operating on flat surfaces. In this work, vibration and small deviations in the floor were not accounted for or corrected. Therefore, this method yielded a slightly less accurate map than using 3D ICP.

3. Three-Dimensional Transformation Method

The second method used in this work utilized the 3D transformations computed from the ICP algorithm. Both methods used the same point cloud and 2D pose information received on the remote desktop computer, and therefore they could both be run on the same data set for comparison.

Normally, ICP scan matching involves calculating a relative transformation between two successive scans, accumulating the relative transformations into an absolute transformation, and finally transforming the point clouds by this absolute transformation. In this case, every tenth scan was received, and although the point clouds had sufficient overlap to prevent gaps in the map, they lacked enough overlap to make ICP reliable. In order to correct this problem, the 3D ICP algorithm was assisted by an initial transform from the 2D pose estimates. Each of the 3D point clouds received by the remote desktop was transformed by the accompanying 2D transformation just as in the previous method.

Each transformed point cloud had the ground plane segmented by fitting a plane with a normal unit vector in the positive Z direction. The outlier points were then stored in a temporary point cloud that was sent to the ICP algorithm. The ICP algorithm returned a transformation similar to (23) but including the smaller rotations about the X and Y axes in the {ROBOT} frame. These relative transformations were accumulated into an absolute transformation for each point cloud that would map them into the {WORLD} frame. The original point clouds (transformed by 2D pose but including ground plane) were then transformed by the 3D transformation calculated from ICP. The resulting transformed 3D point clouds were then merged into a map in the same fashion as the 2D transformation method.

This method created a slightly more accurate map at the cost of speed. Both methods relied on the data transmitted over the ROS network, which was saved to hard disk on the remote computer.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS

This chapter examines the results from several experiments conducted with Robot 1 and Robot 2. Throughout development, many experiments were conducted to test LLAM methods and tune parameters. Included in this chapter is a brief description of the development of the LLAM algorithms for each robot, as well as an examination of the performance of the final version of each algorithm. Localization, map building, and speed are the primary areas of interest here. Additionally, an examination of map interpretability is included. Several conclusions are drawn from the results presented in this chapter, the most important of which are further discussed in Chapter VI.

A. LOCALIZATION

Initially, Robot 1 was localized using 3D point clouds from the Pico monstar sensor. This method provided for localization of the robot assuming six DoF using the iterative closest point method. The ICP algorithm was fast enough to run in real-time at approximately 10 Hz using the mounted full-size laptop. The primary problem encountered was matching degeneration due to the limited field of view and range of the Pico monstar sensor. The sensor, although relatively accurate for the size and cost, is not well suited for mapping applications. The manufacturer advertises it for 3D scanning of objects, gesture recognition, and similar activities in which it is primarily static, looking at objects that are placed intentionally within the FoV. For room scanning and autonomous navigation, the sensor has a relatively limited range and often cannot see far enough to accurately sense walls or other objects in the room. Although the sensor can return points up to six meters, many of those points are of low confidence, or with noise. When filtered, the effective range of the sensor is closer to three meters, depending on a number of environmental factors. In order to avoid matching degeneration, the algorithm was adjusted to use the 2D Hokuyo LIDAR data as a primary means of localization, and the 3D LIDAR data for map building and obstacle avoidance. This combined 2D and 3D implementation was used on the final version of Robot 1 and on Robot 2.

1. Matching Degeneration

In this work, the LLAM algorithm was structured around matching of consecutive scans. As discussed in Chapter III, the iterative closest point method relies on finding correspondences between points and computing the relative transform between point clouds. Therefore, if two point clouds do not contain a sufficient amount of corresponding points, the algorithm either fails to converge, or it returns a transform that maps points incorrectly to the previous point cloud. If the SLAM method involved multiple sensors to estimate pose, the scan matching may be assisted by an initial transform computed from other sensor data. In this case, the central question was to determine if it could be done quickly and efficiently without additional sensors.

An example of matching degeneration is seen in Figure 25. Shown in the figure are two consecutive experiments with Robot 1 executing a 90-degree left turn in the same portion of Spanagel Hall 521. This experiment was conducted without the use of the 2D Hokuyo LIDAR. Additionally, the other navigation methods were disabled as a loss of localization would cause Robot 1 to wander erratically. In this case, the robot was programmed only to travel several meters, turn left, and continue traveling roughly five meters. Between the two experiments, the translational and rotational speeds of the robot were increased in order to cause the robot to outpace the ICP algorithm.

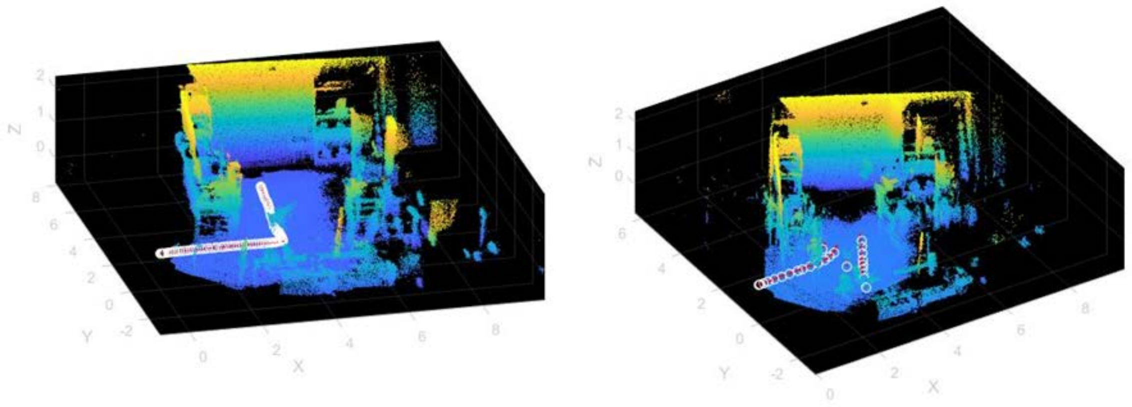


Figure 25. Example of Matching Degeneration

The white circles depict the estimated 3D position and the red vector arrows contained within each circle represent the estimated orientation. In the case on the left, the robot made the corner while accurately localizing using 3D ICP. In the case on the right, the robot was unable to match sequential scans even though the trajectory was nearly identical. It can be seen that the LLAM algorithm lost the ability to both localize the robot accurately and build an accurate map. In the LLAM algorithm developed for this work, once localization is lost it cannot be recovered without loop-closure.

In order to limit the loss of localization associated with the short range of the Pico monstar, the 2D Hokuyo LIDAR was utilized. The Hokuyo LIDAR can return accurate points from anywhere in the laboratory space with a 270° FoV, which makes it far less prone to matching degeneration. A map of the laboratory space created after inclusion of the Hokuyo LIDAR data is seen in Figure 26.

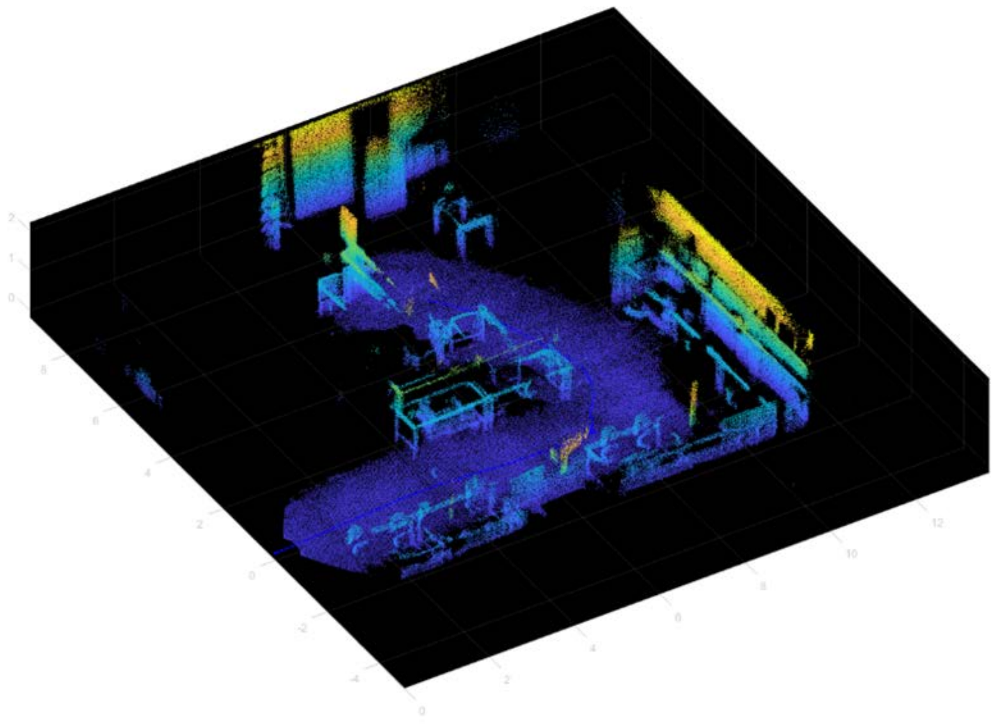


Figure 26. 3D Map of Spanagel 521 Laboratory Space Using 3D and 2D LIDAR Data

It can be seen from Figure 26 that the 3D map closely resembles the interior laboratory space complete with work benches and chairs. The colormap corresponds to height in the vertical dimension with blue depicting zero m (ground plane) in the {WORLD} frame. As the height increases, the colormap transitions to green, then to yellow at the ceiling. The ceiling plane in this figure is removed so that the contents of the room can be viewed. The outline of the work benches is clearly visible. However, because the 3D LIDAR was mounted at a similar height to the workbenches, the planar surface of the benches was not fully visible to the LIDAR. Black corresponds to space that was not mapped by the 3D LIDAR. The 3D portion of the map is limited by the FoV of the Pico monstar. For comparison, an overlaid 2D SLAM map (pink) generated from the Hokuyo LIDAR during the same experiment is seen in Figure 27. Although the robot could sense the entirety of the space using the 2D LIDAR, the limited FoV of the Pico monstar meant that only a portion of it was mapped in three dimensions.

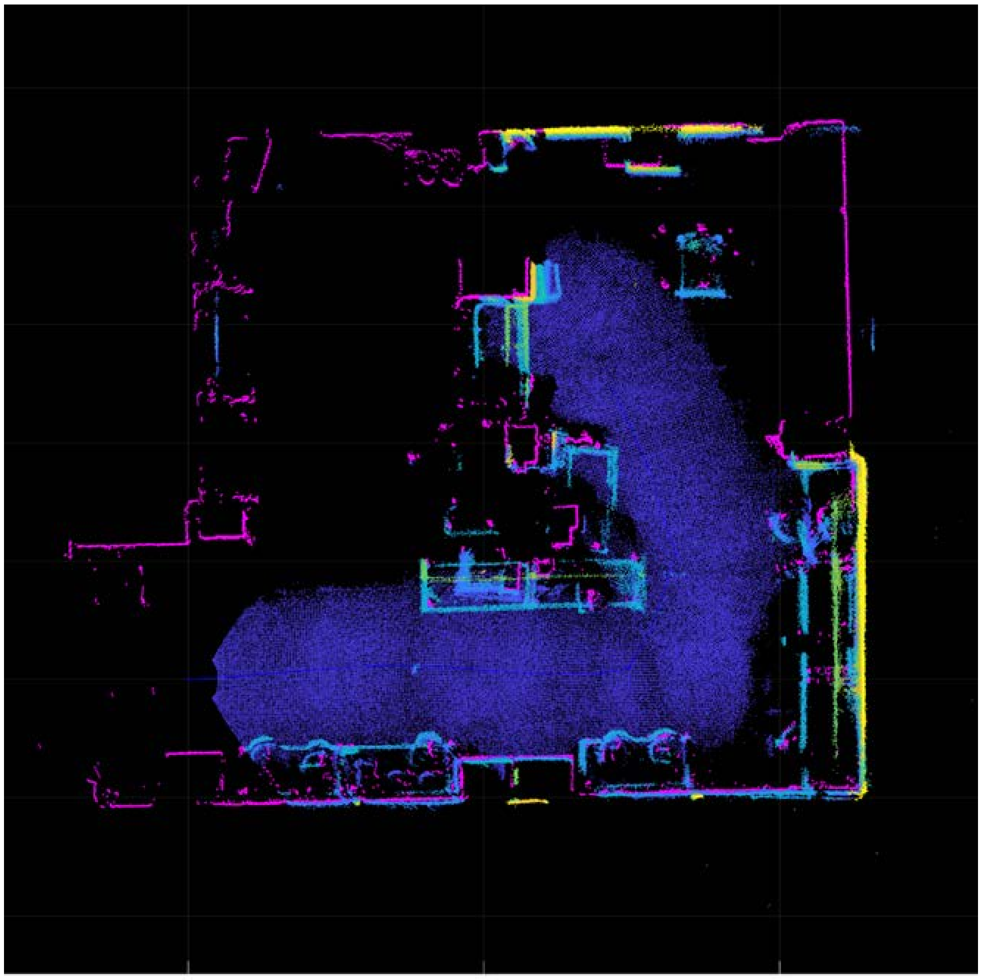


Figure 27. 3D Map of Spanagel 521 Top Down View Overlaid with 2D Hokuyo SLAM Map

In order to build a full 3D map, another more powerful 3D LIDAR must be acquired. Spinning 3D LIDARs, such as the Velodyne used in the KITTI data discussed in Chapter III, are commercially available and necessary for continued development of a 3D autonomous mapping robot. Additionally, point cloud returns from these 3D spinning LIDARS are formatted in laser lines; 2D navigation provided by the Hokuyo in this work, could still be accomplished using a single laser line from a 3D spinning LIDAR.

2. Accuracy on a Closed Loop

As the robot is not equipped with sensors other than the two LIDAR units, determining the accuracy of localization is difficult. From visual inspection of the 2D and 3D maps compared with the environment, it can be seen that the map is relatively accurate. Certainly from a tactical perspective, this map would be useful to an operator.

In order to determine values for drift and localization error, Robot 2 was given a set of waypoints that terminated at the point from which the robot was initialized, ${}^{W}(0,0)$. In this experiment, the robot traveled the intended trajectory, and when it reached the final waypoint ceased movement. From that point, distance to the robot center was measured from a marked point on the floor where the robot was initialized. The last calculated pose estimate was retrieved from the robot memory and compared to the measured point, as seen in Table 2.

Table 2. Closed Loop Test Data

Measured XY Final Position (m)	Estimate XY Final Position (m)	Final Position Error (m)	Trajectory Length (m)	Position Error (%)
${}^{W}(0.56, 0.30)$	${}^{W}(0.2490, 0.1096)$	0.365	17.09	2.14

In this experiment, the robot drifted 2.14%, which is similar to the drift seen in the data output from the KITTI Benchmark test algorithm. The robot was programmed to consider a waypoint reached when it was within 0.30 m of the intended waypoint. This prevented the potential field algorithm from trying to make velocity adjustments that were too small for the robot to execute when near to the goal. As implemented in this work, the drift of the robot does not contribute to a likelihood that it will collide with obstacles. Obstacle are detected, and steering commands are updated every iteration of the LLAM algorithm. Therefore, drift will only affect the generation of an accurate map.

3. Algorithm Speed

During mapping experiments, the speed of the LLAM algorithm was not rate controlled. Instead the algorithm was run at maximum speed. This meant that each iteration of the algorithm varied in duration. Since pose estimates were not being calculated kinematically from other sensors, the elapsed time between laser scans was not important so long as the algorithm ran sufficiently fast to avoid matching degeneration. Consequently, the speed of the algorithm varied greatly from one iteration to the next and from one trajectory to the next. In general, the LLAM algorithm on Robot 2 ran between 5 and 10 Hz, occasionally exceeding 10 Hz, but rarely performing below 5 Hz. This proved an acceptable speed given that the robot forward velocity was limited to 0.4 m/s. The actual speed of the robot was updated every iteration of the LLAM algorithm by the output of the potential field model. In most cases it had a forward velocity of around 0.25 m/s depending on the distance to the goal and the obstacles sensed in the environment.

Transmitting 3D point clouds wirelessly using IEEE 2.4 GHz 802.11n protocol proved to be the largest bottleneck. Prior to constructing Robot 2, an experiment was conducted to transmit 3D point clouds from the Pico monstar sensor attached to the SlimPRO, to the desktop computer for map building. During this experiment, the robot was not moving and none of the navigation or obstacle avoidance algorithms were running. The SlimPRO was only retrieving data from the Pico monstar at a rate of 10 Hz, formatting the ROS message, and sending it over the ROS network. In this case, the algorithm was rate controlled to 10 Hz to ensure that the variable being measured was transmission time over the ROS network. As suspected, 3D point cloud messages could not be transmitted at 10 Hz over the ROS network. Initially, the point clouds were received at roughly 0.5 Hz on the desktop computer. Changing the format of the ROS message, as discussed in Chapter IV, increased this speed to a more consistent 1 Hz.

Actual data rates over the wireless connection averaged between 8 Mbps and 10 Mbps. After removing intensity and depth confidence data, the size of 3D point clouds varied based on the number of laser returns, between roughly 5 Mbit and 10 Mbit. The 802.11n protocol should be able to support much higher data rates than were observed in this experiment, which would allow faster real-time transmission of point cloud data. The actual cause of the slow network data rate is beyond the scope of this work but is important to note for future investigation.

Attempting to solve the transmission delay issue using a buffer was attempted. However, as the rate of collection outpaced the rate of transmission by a factor of 5 to 10, the buffer inevitably filled up. Continuously increasing the size of the buffer only took memory resources away from other elements of the LLAM algorithm. As the maximum speed of the LLAM algorithm was roughly 10 times the rate of transmission, transmitting only every 10th point cloud was the method selected to avoid an excessive backlog.

Without the use of a buffer, occasionally 3D point cloud messages and 2D pose estimates would be transmitted out of sync. One message or the other having been overwritten before transmission. Certainly, an algorithm could be written to run on the robot end to correct this, but it would slow the rate of the LLAM algorithm. Instead, if the LLAM algorithm overwrote either a pose message or point cloud message before transmission, both messages would be transmitted anyway with a frame ID appended to the header. On the remote computer end, the frame IDs would be compared, and mismatched messages would be discarded. This solution had the effect of occasionally recording only the 20th or 30th scan from the robot, but so long as the message frame IDs matched, the 2D pose estimate could be used to transform the 3D point cloud for map making without significant loss to the map.

B. MAP BUILDING

1. Map Accuracy and Interpretability

In general, the accuracy of a map is absolute. The error in the map being the difference between the environment that the map depicts and the environment as it actually exists. For the application of this work, it was decided from the beginning that the absolute accuracy of the map is less important than the speed at which it is acquired. In a tactical setting, such as on-site raid planning, any map that is not misleading is better than no map at all. With that in mind, the goal of this research was not to create a highly detailed map that was accurate within the tolerances of the LIDAR sensors. Instead, it was to determine if LLAM is a suitable method for building a hasty map that can be easily interpreted by a human operator. In that respect, the accuracy of the map was primarily determined by visual inspection. If the map resembled the environment it was compiled to represent and it was interpretable to a person, it was considered a successful employment of LLAM.

The map depicted in Figure 28 was compiled from data collected by Robot 2 during the closed loop experiment. It is important to note that human experimentation was not included in this work. Therefore, the interpretability of the map is subject to a preexisting understanding of the environment. Throughout experimentation the robot had to be placed in the laboratory and it was not allowed to operate unsupervised for safety reasons. Thus, the operator already understood what the environment looked like and could correlate that understanding to the map that the robot produced.

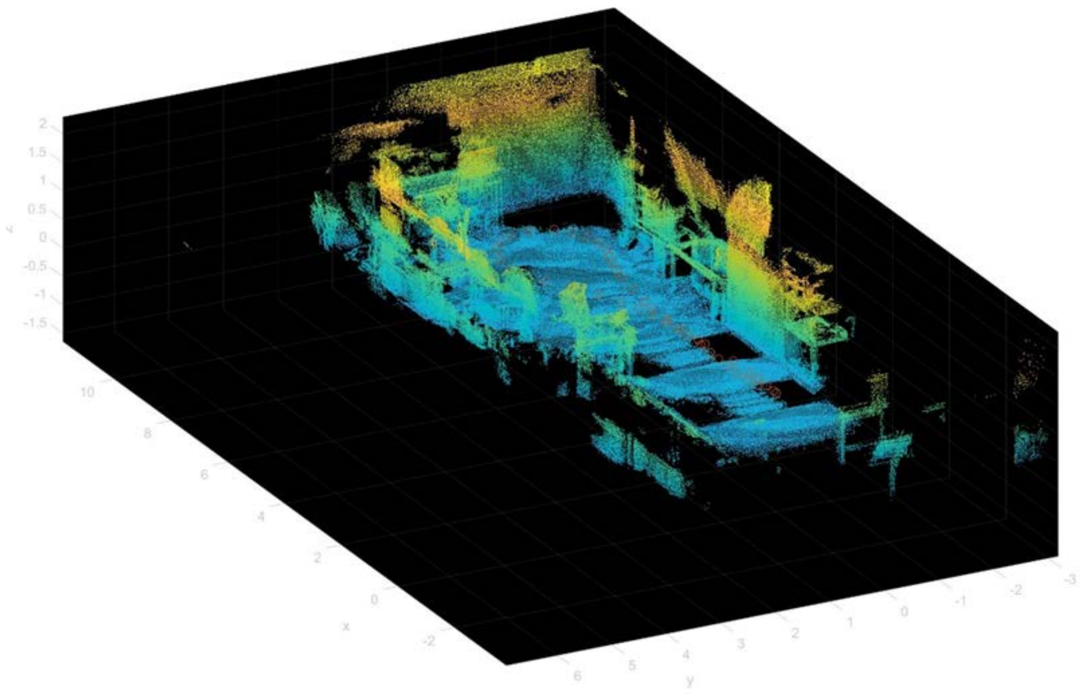


Figure 28. 3D Map of Spanagel 521 from Robot 2 Closed Loop Experiment

Interior spaces, such as those depicted in this chapter, are difficult to view from an exterior perspective. If the map is complete, it includes walls, a ceiling plane, a ground plane, and typically the room is fully enclosed. In order to interpret the map without trying to view the contents of a room through the walls or ceiling, a portion of the map must be removed, or the map program must have the ability to render a view from “inside” the room. In this work, the ceiling plane was removed from all maps so that the contents of the room could be viewed from an exterior perspective. Additionally, the points were color mapped corresponding to distance in the vertical dimension. The maps produced here are still somewhat hard to interpret without a preexisting understanding of the environment. That is partially due to the laboratory equipment present that is not common to most rooms; however, discrete point cloud maps may not be the best solution for an easily interpretable map.

One solution is to create a user interface so that the map can be explored interactively. An example of this is seen in Figure 29, where the point cloud map was loaded into Unreal Engine 4 and compiled into a third person video game. This allows the user to explore the map in a more familiar and interpretable way. The creation of a third person video game type interface was done separately from the LLAM algorithm. It is included for demonstration only, and as a consideration for another avenue for future development. It was not scripted into the LLAM algorithm or interfaced with the robot.

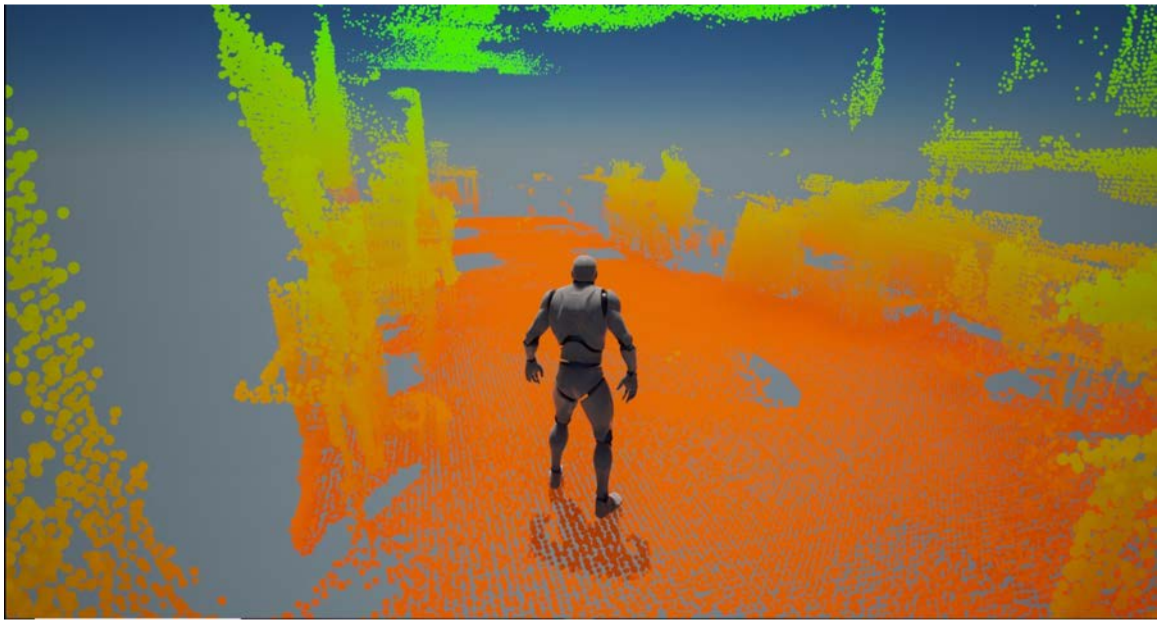


Figure 29. Example of a User Interface that Allows Map Exploration

2. Comparison of 2D and 3D Transformation Methods

As discussed in Chapter IV, the map can be compiled using the 2D transformations computed from NDT scan registration, or the 2D transformations can be used as initial transformations for 3D ICP point cloud registration. For comparison, both methods were used on the data collected from the Robot 2 closed loop experiment. As can be seen in Figure 30, with a ground mobile robot having three DoF, very little difference is detectable in the map. The image on the left was compiled from the 2D NDT transformations, the image on the right was compiled from the 3D ICP transforms. The times to compile the maps were 0.495 s and 6.06 s for the 2D and 3D transformation methods, respectively. Although the 2D transformation method was more than 12 times faster, the duration of six seconds for the 3D method is sufficient to meet the tactical requirements here. Thus, either method works with a robot constrained to three DoF.

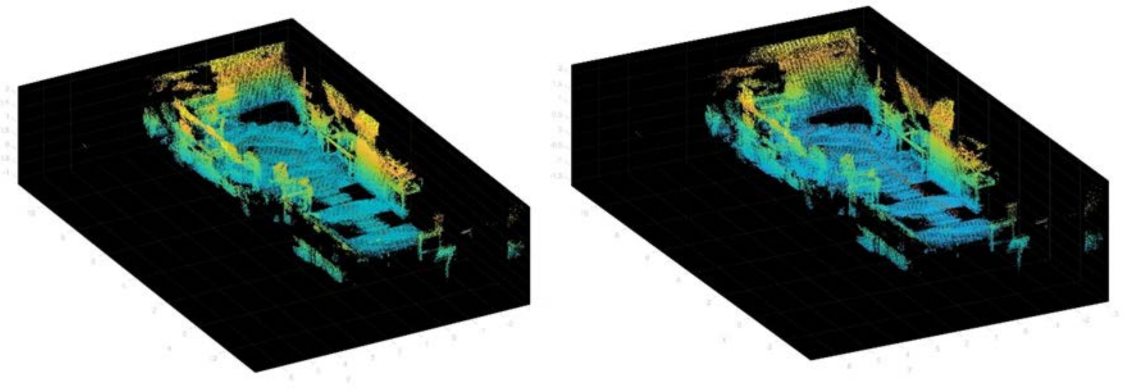


Figure 30. Comparison of 2D (left) and 3D (right) Transformation Methods of Map Building

Careful inspection reveals that the 3D transformation method yields a map with slightly more distinguishable objects. The general accuracy of the maps, however, cannot be differentiated without very close comparison. In either case, objects are difficult to distinguish from a fixed exterior perspective. Thus, an ability to explore the map interactively is necessary, as discussed in the preceding section.

C. OBSTACLE AVOIDANCE AND ROUTE PLANNING

The potential field model of obstacle avoidance proved successful in the laboratory environment when the experiments were constructed to minimize the likelihood of encountering a local minimum. Previous NPS thesis work integrated local minima escape methods that were not included in this work. These methods would have expanded the capability of the robot; however, the potential field model has limitations that may make another obstacle avoidance model preferable for this application. For example, the Hokuyo LIDAR has a fixed angular resolution. At certain distances slim obstacles, such as table legs, may fall in between the laser shots from the Hokuyo LIDAR. Since the 3D LIDAR returns were sorted into angle bins corresponding to the angular resolution of the Hokuyo LIDAR, the 3D returns did not always assist the robot in detecting these obstacles until they were relatively close.

Although the robot did not fail to detect and avoid slim obstacles prior to collision, it frequently detected them at a close range where it could not correct the trajectory without slowing down significantly. Given the 0.25° angular resolution of the Hokuyo, the LIDAR should be able to detect most of the table legs (several centimeters wide) at a range of several meters. In practice however, it frequently did not detect them until it was within one meter.

In general, it is not advantageous to use the maximum range of the Hokuyo LIDAR for potential field calculations. Assigning repulsive forces to obstacles that are very far away creates a more complicated potential field and requires significant tuning of the gains in order to make the robot reach the goal. However, the extended range of the Hokuyo LIDAR, or a 3D spinning LIDAR if equipped, gives the robot the ability to sense obstacles far away. If a different obstacle avoidance model were implemented, such as an occupancy map-based planning algorithm, the robot would be able to plan routes out to the extent of the LIDAR range. Combined with a frontier search algorithm, the robot could then explore an interior space more efficiently. This model, however, would be susceptible to drift in a way that the potential field model, as implemented here, is not. If the pose estimate of the robot drifted from the actual pose significantly, the robot could collide with obstacles. This implementation would require loop-closure or some other method of iterative pose refinement.

In this work, a hybrid model of potential field obstacle avoidance and occupancy map-based planning was implemented on Robot 1. The potential field algorithm ensured that the robot did not collide with obstacles due to drift, and the occupancy map-based Hybrid A* Search planned a route to the next goal. This model allowed the robot to plan a route based on all the known obstacles detected from any previous pose of the robot. It proved effective as a navigation and obstacle avoidance method but suffered from the long pauses required to build the occupancy map, as previously described. The SLAM and occupancy map building algorithms used in this work were included in the MATLAB Computer Vision and Navigation Toolboxes and are not necessarily optimized for speed or autonomous robotics. Programming similar algorithms optimized for speed was not examined in this work, and consequently Hybrid A* Search planning was not included on Robot 2. However, if combined with pose refinement, occupancy map-based planning may prove more flexible and efficient than the potential field model used here.

Furthermore, the constrained degrees of freedom and relatively slow movement of the ground mobile robot allowed the dynamics to be simplified to translational and rotational velocity commands sent to the robot. If LLAM is extended to flying robots, operating at faster speeds, and considering higher order dynamics, the method implemented here will need to be reexamined.

Finally, the Hybrid A* Search algorithm did not have the ability to incrementally re-plan the route while the robot was in motion. If a previously unidentified obstacle was encountered, the robot relied solely on the potential field model to avoid it. This made it susceptible to the local minima problem. If an escape method was implemented, the robot would still need to stop and re-plan the route once the local minimum was escaped. D* Search for example, includes the ability to incrementally re-plan a route using a similar occupancy grid and cost map approach [22]. Implementation using D* Search, or another similar method, could alleviate susceptibility to the local minima problem and remove the need for escape methods.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

This thesis research developed two robots capable of localizing and mapping using LIDAR sensors only. Each robot operated differently providing valuable insights into LLAM as a means for tactical mapping, as well as future development required to make this a feasible military capability. This chapter includes conclusions drawn from experimentation and recommendations for future work.

A. ASSESSMENT OF GOALS

1. Feasibility of LLAM for Autonomous Interior Mapping

LIDAR localization and mapping is a feasible method of generating three-dimensional interior maps in a tactical setting. It can be performed in real-time with current mobile computing resources. With further development, LLAM could be implemented in military operations successfully. Significant further development is needed to make this a technology capable of fielding. However, the concept is sound based on current computer technology, and advances in computer technology will make it even more feasible in the future.

The inability of LIDAR to detect transparent windows is an issue that may make a purely LLAM enabled robot insufficient for tactical operations. In this work, none of the windows in the test spaces extended to the floor. A flying robot, however, would not be able to detect the windows solely with LIDAR and would likely try to fly through windows, perceiving them as unoccupied space, and the area beyond them as unexplored space. An additional sensor may be required to prevent this.

Many commercially available LIDAR sensors now include an IMU built into the LIDAR unit itself. As the IMU is already a component of the LIDAR, future research in LIDAR mapping should seek to use IMU data to assist in point cloud registration and pose estimation. The initial assertion in this thesis was that the inclusion of additional sensors increases weight, complexity, power requirements, and data processing requirements. This makes additional sensors less than desirable for tactical applications. As the commercial LIDAR market begins including IMUs, and in some cases color imagery, into a single sensor of similar size and weight, it only makes sense to utilize the provided data for improved localization and mapping.

2. Adequate Resolution

Determining an adequate resolution for useable tactical maps is both subjective and variable based on the environment. Throughout experimentation, the majority of final maps were downsampled using a three cm voxel grid filter. This parameter was adjusted through experimentation. The resulting resolution was sufficient to build an interpretable map. In most cases, common objects such as computer monitors, chairs, and work benches were easily distinguishable. Other, less common objects, such as torsion plants and other robots, were difficult to distinguish. In most cases, increasing map resolution assisted in identifying these less-familiar objects. However, creating extremely dense point clouds made the maps less interpretable. Partial transparency in the map creates contrast, which makes the objects in the scene more visible. Thus, a point exists where increasing point cloud density decreases map interpretability. Through experimentation, the aforementioned values were determined to be optimal for the test spaces. In a more common environment, such as a house, a lower resolution may be perfectly acceptable to distinguish common appliances and household items.

The gradient colormap used to distinguish elevation in the processed maps was acceptable. However, when viewed from an exterior perspective, it can be difficult to differentiate small objects. For example, an object on a table could be difficult to distinguish from the table itself. This is because the color value was based on elevation, and because the object and table had similar elevations, they had similar colors as well. If the map was rendered with a light source that cast shadows, such as in Figure 29, objects may be easier to distinguish. Creating a mesh-based map from the point cloud may also be a suitable solution.

3. Obstacle Detection and Autonomous Navigation

Robot 1 combined LLAM with obstacle detection and navigation effectively to a certain degree. The robot required long pauses at each waypoint to compile the SLAM and occupancy maps, as well as run the frontier search and Hybrid A* Search algorithms. The frontier search and Hyrbird A* Search algorithms contributed negligibly to those pauses. Those algorithms could likely be performed on the move if the required occupancy map input could also be built on the move. This is an area that requires further development. However, there is no evidence to suggest that the occupancy map process cannot be optimized for speed and incorporated in real-time. This further development is a necessary requirement for tactical employment.

4. Map Visualization Method

A color-mapped discrete point cloud provides a map that is acceptable but suboptimal. It is certainly better than no map at all. However, it does not depict an environment as humans see it. Several commercially available LIDARs have the ability to assign an RGB color to each returned point. This allows the discrete point cloud to be colorized in a similar way to an image and makes the map much easier to interpret.

As already discussed, an exterior viewpoint creates a problem where a portion of a room needs to be removed in order to discern what is inside. In an experimental environment, it is fairly simple to determine which wall or ceiling plane to remove in order to view the interior. In a totally unknown structure, it would likely be much more difficult to understand. For example, if the robot were to map the entirety of a multiple story building with interior spaces that did not border exterior walls, the map would need to be partitioned multiple times to view all of the space from an exterior perspective. In this respect, an interactive map would be far more useful. For many years, the video game industry has been creating explorable, easily interpretable 3D maps. Additionally, nearly all service-age adults grew up in a time when 3D video games were widely accessible. A similar map interface to video games would be a logical and easily achievable method for visualizing 3D maps.

B. LIMITATIONS

1. Hardware

Robot 1 and Robot 2 were both constructed with materials already on hand within the ECE Control Systems laboratory. Although these materials proved sufficient, none of them are on the cutting edge of technology, and they have a number of limitations that newer technologies do not. First, the PMD Pico monstar LIDAR is not ideal for autonomous interior mapping as it has a short range and limited field of view. A 3D spinning LIDAR would provide longer range, a larger FoV, and more accurate localization and mapping. It would also remove the reliance on the 2D Hokuyo LIDAR for navigation.

Similarly, the full-size laptop mounted on Robot 1 could perform six DoF LLAM in real-time but struggled to build an occupancy map and plan a route fast enough for tactical applications. Since the SlimPRO had less computing power than the laptop, a simplified three DoF localization method was used on Robot 2. Additionally, the Hybrid A* Search and frontier search algorithms were not included. Neither the laptop nor the SlimPRO represent what is currently commercially available in terms of performance. A more powerful on-board computer would be beneficial to future LLAM research and development.

Wireless transmission of point cloud data was a significant bottleneck. As it was necessary for other devices to be using the laboratory wireless router, it is possible that bandwidth was limited by other experiments and network traffic. Future research should attempt to identify a suitable network protocol for wireless point cloud transmission. Perhaps newer IEEE 802.11xx protocols, new router hardware, or creation of an Ad Hoc network would solve this problem. The proliferation of 5G technology may provide another avenue that is worth examining.

2. Software

MATLAB is an exceptionally useful tool for prototyping; however, it is not optimized for speed. In fact, MATLAB has a built-in coder tool to convert MATLAB code to C or C++ to speed up slow algorithms [36]. MATLAB was the language selected for this work for several reasons. First, MATLAB was used for previous NPS theses on the topic of autonomous robotics. A portion of the code written for those works could be easily adapted for use here. Second, MATLAB includes numerous functions and toolboxes that assist in development and analysis. Finally, MATLAB allows for the integration of Windows OS computers with ROS, which was beneficial in the development of Robot 2.

Since LLAM requires significant computational power and memory resources, another programming language, such as C or C++, is preferable. Additionally, these languages allow specific processes to be multithreaded. In [11], Lin and Zhang divided a received point cloud into three subframes and matched them independently with the global map, each through a dedicated CPU core. The Loam_livox algorithm in [11] was written in C++ and designed specifically to operate on multicore processors, achieving a real-time speed of 20 Hz.

The LLAM algorithm developed here can be further optimized for speed while still running in the MATLAB environment. However, various open source libraries for point cloud processing already exist. For Example, Point Cloud Library (PCL) is a free C++ library of point cloud and image processing algorithms, many of which are similar to those used in this work. It includes algorithms for filters, registration, KD tree/octree construction, feature extraction, etc. [37]. According to Rusu and Cousins, PCL is written for efficiency and performance on modern CPUs, and it is also integrated with ROS [37]. Point Cloud Library, or other similar C/C++ based projects, could provide a better development environment than MATLAB for future work in LLAM.

C. FUTURE WORK

Simultaneous localization and mapping is a continuously evolving field with significant advances occurring frequently. LIDAR based SLAM is a newer branch of the SLAM problem with significant space for future work. Included here are recommendations for future work utilizing the LLAM algorithm. These areas for future work are not all encompassing but are focused on the practical military applications of LLAM.

1. Algorithm Optimization and Hardware Upgrade

Upgrading the robot used in this thesis by incorporating a spinning 3D LIDAR and removing the 2D Hokuyo LIDAR is a logical next step. The methods contained herein should work with a 3D spinning LIDAR; however, this would require significant rewriting of the code included in the appendices. Additionally, much effort was taken to optimize the LLAM algorithm and included functions. Certainly, the algorithm can still be improved by converting functions to C++, experimenting with function ordering, etc.

Due to the slow movement speed of the robot, the effects of motion blur within a 3D point cloud were considered negligible. Upgrading the equipment and improving the speed of the algorithm would allow the robot to move faster and will require a method to compensate for motion blur.

Loop-closure was attempted but found unnecessary and removed. The small-scale nature of the map environments combined with a generally one-way path meant that the robot was unlikely to revisit a specific location. In many cases, such as travelling down a hallway, it is obvious from the pose estimate and trajectory that the robot is unlikely to close a loop. In these cases, continuously running loop-closure queries wastes computational power. However, loop-closure allows for drift correction and pose refinement. Future work should develop a method to search for loop-closures triggered by specific circumstances. For example, a loop-closure query may be executed when the robot exits a room from the same door it used to enter.

2. Point Cloud Registration

This work utilized a successive scan matching method based on iterative closest point and normal distribution transform methods. Many other techniques exist that were not examined here, some of which may be superior for this application. Additionally, a new, novel method of point cloud registration could be developed specifically for the small scale, high speed, tactical nature of the problem here. The combination of neural networks or other machine learning algorithms and point cloud data would be similarly useful for this application. For instance, semantic segmentation could be used to identify both obstacles to the robot, and threats to the operators who will presumably enter the building. It could also be used to segment dynamic obstacles, which was a limitation of the LLAM algorithm and occasionally caused matching degeneration.

3. Obstacle Avoidance and Navigation Model

In this work, the limitations on autonomy were primarily due to the models selected for obstacle avoidance and navigation. Robot 1 could function nearly autonomously but at a slow speed. Robot 2 required pre-planned waypoints, but it operated without stopping and could transmit point clouds wirelessly. Tactical employment of a similar robot would require characteristics of both Robot 1 and Robot 2. The Robot 1 LLAM algorithm utilized a hybrid potential field and occupancy map model for obstacle avoidance and navigation. The potential field calculated steering commands to avoid obstacles, and the occupancy map informed the route planning and frontier search algorithms. Future work should seek to develop a combined model for navigation and obstacle avoidance that can incrementally re-plan the route while in motion.

4. Map Building and Human Interactivity

The interpretability of maps presented here was subject to a preexisting understanding of the environment being mapped. Although less technical in nature, in order to develop this technology as a fieldable capability, further research needs to be conducted on human subjects. Human experimentation should include methods for visualizing the map as well as user input interfaces. Additionally, in combat operations a threshold exists where the size, weight, and complexity of a system can become as important as the capabilities of the system. That is, if the system is large, heavy, or difficult to use, it can become a distraction from the mission at hand. In these cases, typically the system is left behind. In the worst cases, distraction from the situation can endanger the operator or military unit. Careful consideration, supported by human subject research, must be given to the usability of a tactical interior mapping robot. Such a robot must meet the needs of the military operator without becoming burdensome.

5. Adaptation to a Flying Robot

For tactical purposes, a ground mobile robot has obvious limitations. A flying robot, such as a quadcopter drone, can avoid ground obstacles easily and seems to be a preferable solution. Additionally, a flying robot could provide a more complete map than a ground mobile robot as it can change elevation to inspect the top of objects that are not be visible from the ground. A six-DoF model similar to that developed for Robot 1, could account for the pitch, roll, and vertical translation of a flying robot. However, research needs to be conducted into the dynamics of a flying robot and how those dynamics effect LLAM. In this work, the slow speeds of the robot allowed the dynamics to be approximated as a series of step inputs. If the robot changed speed, it did so relatively quickly and the acceleration was neglected. This simplified model will not likely work for a flying robot.

APPENDIX A. ROBOT 2

A. MASTER.M

```
%Master.m collects data, runs the obstacle avoidance and navigation
%scripts, and transmits 2D poses and 3D point clouds via the ROS
network
%to the remote computer running Listener.m. This script should be
%initialized via SSH from the remote computer after launching the
%ROS Core and Hokuyo node from the terminal.
%Instructions are included in Listener.m.

clearvars; clc; close all;
addpath('/home/ecejames/Jameson_Thesis/myMatlab')

%% Abort Callback used to break loop if the robot malfunctions
%This only functions if the GUI is running  on MobaXTerm.
%To abort: Press 'a' on the keyboard.

stringInput = 'x';
h_fig =figure;

%% Initialize PMD Pico Monstar
cameraDevice = initializePMD();

%% Connect to Robot
p3_connector_Payne('/dev/ttyUSB0');
disp('Connected to Robot');
pause(0.8);

%% Initialize Nodes, Subscribers, and Publishers,
% and wait for other Machine to connect

%Initialize node
rosinit('localhost','NodeName','slimPRO')

%Initialize Publishers and Subscribers
PMDPub = rospublisher('/scan3D','sensor_msgs/Image');
PMDSub = rossubscriber('/scan3D','sensor_msgs/Image');
HkPub = rospublisher('/scan','sensor_msgs/LaserScan');
HkSub = rossubscriber('/scan','sensor_msgs/LaserScan');
RobotPosePub = rospublisher('/pose','geometry_msgs/PoseStamped');


%Wait for remote machine to connect to ROS network.
disp('Waiting for Other Machine to Connect...')
while true
    ls = rosnode('list');
    if any(contains(ls,'/windows_machine'))
        disp('Other Machine Connected')
        break %If rosnode list contains /windows_machine, break
    end
end

%% Initialize script variables and containers
%Variables
rng(0);
navMode = 0;
maxScans = 1000;

%Preplanned goals (# of Sp521 floor tiles)
goalMat = [11 -1; 13 0; 12 2; 10 2; 0 0];
goalMat = goalMat.*610;     %Convert floor tile distance to mm
sz = size(goalMat);
numWaypoints =sz(1);
minDepthConfidence = 155;
hkOffset =-0.1;
PMDOffset =-0.16;
PMD_height =0.435;
R = [ 0 -1 0; 0 0 -1; 1 0 0 ];
PMDtrans = rigid3d(R,[PMDOffset,0,PMD_height]);

%Navigaion parameters
global goalTolerance
goalTolerance = 300;
transVel = 100;     %Initial translational velocity for frame 1
wayPointInc = 1;
waypointCurrent = goalMat(wayPointInc,:);
wayPointReached = false;



%Containers and Objects
PMDScans = cell(maxScans,1);
HkScans = cell(maxScans,1);
Hkpgraph = poseGraph;



%% Run loop for collection and transmission of data,
% navigation and obstacle avoidance
pause(8);

i = 1;
while (i < maxScans) && (p3_getBumpersClear)
    ratetic = tic;

    %Retrieve Hokuyo Scans, store as laser scan, and ROSMsg
    Hmsg = receive(HkSub);
    hkCart = readCartesian(Hmsg);
    hkCart = hkCart + [hkOffset 0];
    HkScans{i} = lidarScan(hkCart);

    if i == 1
        %Retrieve, Preprocess, PointCloud Object, and ROSmsg PMD data.
        PMDdata = cameraDevice.getData();
        [PMDCloud,PMDImage] =
processPMD(PMDdata,i,PMDPub,minDepthConfidence,PMDtrans);
        % PMDScans{i} = PMDCloud; %uncomment to store point clouds in
        % SlimPRO memory (optional)
        PoseMsg = createROSpose([0 0 0],i,RobotPosePub);
        send(PMDPub,PMDImage)
        send(RobotPosePub,PoseMsg)
        disp('sent')
        p3_setTransVel(transVel);

        %Run every iteration that is not evenly divisible by 10, this
allows
        %the PMDprocess function to skip computations if the frame
        %is not going to be transmitted.
    elseif ~rem(i,10) == 0
        %Retrieve, Preprocess, PointCloud Object, and ROSmsg PMD data.
        PMDdata = cameraDevice.getData();
        [PMDCloud] =
processPMD(PMDdata,i,PMDPub,minDepthConfidence,PMDtrans);
        %PMDScans{i} = PMDCloud;

        %Get Relative Pose
        relPose = matchScans(HkScans{i},HkScans{i-1});

        %Add relative pose to pgraph and retrieve world pose for
navigation
        addRelativePose(Hkpgraph,relPose);
        poseList = nodes(Hkpgraph);
        poseCurrent = poseList(end,:);

        %Otherwise, send every tenth scan and pose to the desktop.
    elseif rem(i,10) == 0
        %Retrieve, Preprocess, PointCloud Object, and ROSmsg PMD data.
        PMDdata = cameraDevice.getData();
        [PMDCloud,PMDImage] =
processPMD(PMDdata,i,PMDPub,minDepthConfidence,PMDtrans);
        %PMDScans{i} = PMDCloud;

        %Get Relative Pose
        relPose = matchScans(HkScans{i},HkScans{i-1});

        %Add relative pose to pgraph and retrieve world pose for
navigation
        addRelativePose(Hkpgraph,relPose);
        poseList = nodes(Hkpgraph);
        poseCurrent = poseList(end,:);

        %Create ROS pose message
        PoseMsg = createROSpose(poseCurrent,i,RobotPosePub);
        send(PMDPub,PMDImage)
        send(RobotPosePub,PoseMsg)
        disp('sent')
    end
    if i > 1
        %Obstacle Detection and navigation

[fwdVel,rotVel,Frep_r,Fatt_r]=potentialField(poseCurrent,PMDCloud,HkSca
ns{i},waypointCurrent,navMode);
        p3_setTransVel(fwdVel);
        p3_setRotVel(rotVel);

        %Check if waypoint Reached
        wayPointReached = p3_goalReached(poseCurrent,waypointCurrent);
        if (wayPointReached == true)
            if (wayPointInc < numWaypoints)
                wayPointInc = wayPointInc + 1;
            else
                p3_setTransVel(0);
                p3_setRotVel(0);
                break
            end
        end
    end
    waypointCurrent = goalMat(wayPointInc,:);

    t = toc(ratetic);
    ratestr = sprintf('Frame = %i,  Rate = %.3f (Hz)',i,1/t);
    disp(ratestr);
    i = i+1;

    %Check for Abort
    drawnow
    set(h_fig,'KeyPressFcn',@(H,E)
assignin('base','stringInput',E.Key));
    if strcmp(stringInput,'a')
        disp('Aborted');
        break
    end

    %Check to ensure remote computer is connected
    ls = rosnode('list');
    if ~any(contains(ls,'/windows_machine'))
        break %break if /windows_machine disconnects
    end
end %while loop

%% Disconnect PMD Pico Monstar

close all
cameraDevice.stopCapture();
disp('PMD Pico Monstar Disconnected');

%Close serial connections and destroy objects
shutDownAll;
clear PMDPub PMDSub HkPub HkSub RobotPosePub cameraDevice
save('P3_data.mat')
```

```
%% Supporting Functions
function poseMsg = createROSpose(poseCurrent,i,RobotPosePub)
%This function creates a pose message from the Hokuyo LIDAR data
HkHeight = 0.33;
poseMsg = rosmessage(RobotPosePub);
poseMsg.Pose.Position.X = poseCurrent(1);
poseMsg.Pose.Position.Y = poseCurrent(2);
poseMsg.Pose.Position.Z = HkHeight;
quat = eul2quat([poseCurrent(3) 0 0]);
poseMsg.Pose.Orientation.W = quat(1);
poseMsg.Pose.Orientation.X = quat(2);
poseMsg.Pose.Orientation.Y = quat(3);
poseMsg.Pose.Orientation.Z = quat(4);
poseMsg.Header.FrameId = num2str(i);
end

function [PMDCloud,PMDImage] =
processPMD(PMDdata,i,PMDPub,minDepthConfidence,PMDtrans)
%This function preprocesses 3D LIDAR data removing invalid points. If
there
%is one output argument it only processes the lidar data, if there are
two
%output arguments it also creates the ROS image message for
transmission to
%the remote computer.
switch nargout
    case 1
        %Preprocess 3D lidar data
        x = PMDdata.x; y=PMDdata.y; z=PMDdata.z; depth =
single(PMDdata.depthConfidence);
        pointsToKeep = depth > minDepthConfidence;
        x = x(pointsToKeep); y = y(pointsToKeep); z = z(pointsToKeep);
        organizedPoints = cat(3,x,y,z);

        %Rotate and translate point cloud to robot frame
        PMDCloud = pointCloud(organizedPoints);
        PMDCloud = pctransform(PMDCloud,PMDtrans);
    case 2
        %Preprocess 3D lidar data
        x = PMDdata.x; y=PMDdata.y; z=PMDdata.z; depth =
single(PMDdata.depthConfidence);
        pointsToKeep = depth > minDepthConfidence;
        x = x(pointsToKeep); y = y(pointsToKeep); z = z(pointsToKeep);
        organizedPoints = cat(3,x,y,z);

        %Write ROS message
        PMDImage = rosmessage(PMDPub);
        PMDImage.Encoding = '32fc3'; %Single floating point 3 channel
image
        writeImage(PMDImage,organizedPoints);
        PMDImage.Header.FrameId = num2str(i);

        %Rotate and translate point cloud to robot frame
        PMDCloud = pointCloud(organizedPoints);
        PMDCloud = pctransform(PMDCloud,PMDtrans);
end
end



function goalReached = p3_goalReached(poseCurrent,goal)
%This function determines if the robot has reached the current goal.
global goalTolerance

%Pull Data from Current Pose and store in local variables
worldXCurrent=poseCurrent(1)*1000;
worldYCurrent=poseCurrent(2)*1000;
worldXGoal=goal(1);
worldYGoal=goal(2);

%Calculate Distance to Target Point
worldDist=sqrt((worldXCurrent-worldXGoal)^2+(worldYCurrent-
worldYGoal)^2);

if worldDist>goalTolerance
    goalReached = false;
else
    goalReached = true;
end

end

function cameraDevice = initializePMD()
%This function initializes the PMD Pico monstar LIDAR. It is adapted
from
%the manufacturers MATLAB SDK example scripts and written as a local
%function. Through experimentation the preset "use case 2" provided
similar results to
%those achieved varying the camera parameters individually to perceive
the
%lab environement. Other use cases are detailed in the PMD Pico monstar
%documentation. Parameters can also be set individually using the
object
%methods (not documented for MATLAB, see classdef provided in SDK),
however the preset use cases are
%recommended.

   % retrieve royale version information
    royaleVersion = royale.getVersion();
    fprintf('* royale version: %s\n',royaleVersion);

    % the camera manager will query for a connected camera
    manager = royale.CameraManager();
    camlist = manager.getConnectedCameraList();

    fprintf('* Cameras found: %d\n',numel(camlist));
    cellfun(@(cameraId)...
        fprintf('    %s\n',cameraId),...
        camlist);

    if (~isempty(camlist))
        % this represents the main camera device object
        cameraDevice = manager.createCamera(camlist{1});
    else
        error(['Please make sure that a supported camera is plugged in,
all drivers are ',...
            'installed, and you have proper USB permission']);
    end

    % the camera device is now available and CameraManager can be
deallocated here
    delete(manager);

    % IMPORTANT: call the initialize method before working with the
camera device
    cameraDevice.initialize();

    % retrieve valid use cases
    UseCases=cameraDevice.getUseCases();
    fprintf('Use cases: %d\n',numel(UseCases));
    fprintf('    %s\n',UseCases{:});
    fprintf('====================================\n');
    if (numel(UseCases) == 0)
        error('No use case available');
    end
    UseCase=UseCases{2};
    cameraDevice.setUseCase(UseCase);
    cameraDevice.startCapture();
end
```

B. LISTENER.M

```
%Listener.m establishes a ROS node on a remote windows computer and
%subscribes to ROS image messages (PMD 3D point clouds), as well as ROS
%pose messages sent from Master.m running on the robot on-board
computer.
%This script collect data and saves it to a .mat file to be processed
with
%one of the map building scripts.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%Insructions:
% 1. Open MobaXterm or PuTTY and connect to 192.168.0.9
username:ecejames
% 2. $tmux
% 3. $roscore ctrl+b then %
% 4. $rosrun urg_node urg_node ctrl+b then %
% 5. $matlab -nodisplay -nosplash -nodesktop
% 6. >>Master.m
% 7. Run this script in MATLAB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

clearvars; clc; close all

%Preallocate cell arrays for messages
maxFrames = 1000;
PMDMessages= cell(maxFrames,1);
poseMessages=cell(maxFrames,1);

%Connect to ROS master already running on SlimPRO
rosinit('192.168.0.9','NodeName','/windows_machine');

%Initialize subscribers
PMDSub = rossubscriber('/scan3D','sensor_msgs/Image','BufferSize',3);
RobotPoseSub =
rossubscriber('/pose','geometry_msgs/PoseStamped','BufferSize',11);

%Ensure all nodes are active
disp('ROS Node List:')
rosnode list
disp('ROS Topic List:')
rostopic list

%Run while loop until robot stops sending messages
i=1;    %increment
while true
    rateTimer = tic;
    try
        waitTimer = tic;

        %Receive ROS Messages
        poseMsg = receive(RobotPoseSub,30);
        PMDMsg = receive(PMDSub,30);

        %Extract Frame ID from header
        poseFrame = str2num(poseMsg.Header.FrameId);
        PMDFrame = str2num(PMDMsg.Header.FrameId);
        tWait = toc(waitTimer);
    catch ME
        %Error handling to prevent return if timeout occurs
        disp('ROS Message Timeout')
        if ~strcmp(ME.identifier,'MATLAB:undefinedVarOrClass')
            disp(ME.message);
        end
        break
    end

    %Store Messages in cell array indexed by frame ID
    PMDMessages{PMDFrame} = PMDMsg;
    poseMessages{poseFrame} = poseMsg;
    %Print incoming data rate (Hz)
    t=toc(rateTimer);
    ratestr = fprintf('Frame = %i, Rate = %f (Hz), ROS Wait Time = %f
(s)\n',i,1/t,tWait);
    i = i+1;
end

%Disconnect from ROS network
rosshutdown
disp('ROS Shutdown');
clear PMDPub PMDSub RobotPoseSub

%If messages were received save in .mat file.
if ~isempty(PMDMessages) && ~isempty(poseMessages)
    save('P3_output.mat');
end
```

C. MAPBUILDER.M

```
%mapBuilder.m builds a 3D map from the messages stored from Listener.m.
%This script uses the 2D transformation method (3 DoF) and does not
%register point clouds using 3D ICP.

%% Check if workspace is loaded
clearvars;
if ~exist('PMDMessages','var') || ~exist('poseMessages','var')
    load('P3_output.mat');
end

%% Initialize Variables and Containters
vSet = pcviewset;
PMD_height = 0.435;
PMDOffset = -0.16;
R = rotx(90)*rotz(90);
trans = [PMDOffset 0 PMD_height];
```

```
PMDtrans = rigid3d(R,trans);

%% Find and remove empty cells from saved messages

%Find Empty cells
PMDidx = find(cellfun(@isempty,PMDMessages));
poseidx = find(cellfun(@isempty,poseMessages));

%Remove empty cells
PMDMessages(PMDidx) = [];
poseMessages(poseidx) =[];

%% Remove Frames with mismatched Ids

%Extract PMD frame ids and save in numeric vector
PMDFrameIds = [];
for i = 1:length(PMDMessages)
    PMDFrameIds(i) = str2num(PMDMessages{i}.Header.FrameId);
end

%Extract pose frame ids and save in numeric vector
poseFrameIds =[];
for i = 1:length(PMDMessages)
    poseFrameIds(i) = str2num(poseMessages{i}.Header.FrameId);
end

%Compare and adjust frame id vectors so they are the same.
poseFrameIds = poseFrameIds(ismember(poseFrameIds,PMDFrameIds));
PMDFrameIds = PMDFrameIds(ismember(PMDFrameIds,poseFrameIds));

%Remove frames from cell array that do not match
updateLength = length(PMDFrameIds);
for i = 1:updateLength
    localFrameId = str2num(PMDMessages{i}.Header.FrameId);
    if ~any(PMDFrameIds == localFrameId)
        PMDMessages(i) = [];
    end
    updateLength=length(PMDMessages);
end
for i = 1:updateLength
    localFrameId = str2num(poseMessages{i}.Header.FrameId);
    if any(poseFrameIds == localFrameId)
    else
        poseMessages(i) = [];
    end
    updateLength=length(poseMessages);
end




%% Read remaining messages with matching frame Ids
idx = length(PMDMessages);
```

```
ptCloud = cell(idx,1);
wPose2D = NaN(idx,3);

for k = 1:idx
    %Read PMD messages and rotate to world coordinate frame.
    PMDMsg = PMDMessages{k};
    PMDImage = readImage(PMDMsg);
    PMDCloud = pointCloud(PMDImage(:,:,1:3));
    PMDCloud = pctransform(PMDCloud,PMDtrans);
    ptCloud{k} = PMDCloud;

    %Convert quat Pose to 2D Pose and store
    poseMsg = poseMessages{k};
    Eul = quat2eul([poseMsg.Pose.Orientation.W
poseMsg.Pose.Orientation.X poseMsg.Pose.Orientation.Y
poseMsg.Pose.Orientation.Z]);
    theta = Eul(1);
    absPose2D = [poseMsg.Pose.Position.X poseMsg.Pose.Position.Y
theta];
    wPose2D(k,:) = absPose2D;
end



%% Compute transforms from 2D data and add to viewset object
tic
absTform = rigid3d;
for i = 1: length(ptCloud)
    trans = [wPose2D(i,1) wPose2D(i,2) PMD_height];
    rot = rotz(-rad2deg(wPose2D(i,3)));
    absTform = rigid3d(rot,trans);
    vSet = addView(vSet,i,absTform,"PointCloud",ptCloud{i});

end

%% Build Map and view
gridSize =0.03;
ptCloudMap = helperBuildMapFromViewset(vSet, gridSize);
toc
figure
pcshow(pcdenoise(ptCloudMap))
hold on
scatter3(wPose2D(:,1),wPose2D(:,2),PMD_height*ones(length(wPose2D),1));
hold off
xlabel('x');ylabel('y');zlabel('z');

%% Optional: Save Data
saveInput = input('Save Data (y/n)?\n','s');
if strcmp(saveInput,'y') || strcmp(saveInput,'Y')
    name = input('Enter File Name:\n','s');
    fileName = sprintf('%s.mat',name);
    save(fileName);
end
```

```
%% Supporting Functions


function ptCloudMap = helperBuildMapFromViewset(vSet, gridSize)
%This function is adapted from a function included in MATLAB R2020a
example
%BuildAMapFromLidarDataUsingSLAMExample.m. It utilizes point cloud
viewsets
%instead of the pcmerge.m function included in earlier work.

numViews = vSet.NumViews;

% Extract point cloud views and absolute transformations from view set.
ptClouds  = vSet.Views.PointCloud;
absTforms = vSet.Views.AbsolutePose;

% Make point clouds unorganized
for n = 1 : numViews
    ptClouds(n) = removeInvalidPoints(ptClouds(n));
end

% Preallocate map points
totalNumPoints = sum([ptClouds.Count]);
mapPoints = zeros(totalNumPoints, 3, 'like', ptClouds(1).Location);

pointIndex = 1;
for n = 1 : numViews
    % Transform points to reference frame of first point cloud
    ptCloud = pctransform(ptClouds(n), absTforms(n));

    % Accumulate map points
    count = ptCloud.Count;
    mapPoints(pointIndex:pointIndex+count-1, :) = ptCloud.Location;

    pointIndex = pointIndex + count;
end

% Downsample points using voxel grid filter to the requested resolution
ptCloudMap = pcdownsample(pointCloud(mapPoints), 'gridAverage',
gridSize);
end
```

D. MAPBUILDER3D.M

```
 %mapBuilder3d.m builds a 3D map from the messages stored from
Listener.m.
%This script uses the 3D transformation method (6 DoF). First the
%2D pose is used to initially transform each point cloud. Then 3D ICP
%registration is used to calculate another transform to refine the map.




%% Check if workspace is loaded
clearvars;
if ~exist('PMDMessages','var') || ~exist('poseMessages','var')
    load('P3_output.mat');
end

%% Initialize Variables and Containters
vSet = pcviewset;
PMD_height = 0.435;
PMDOffset = -0.16;
R = rotx(90)*rotz(90);
trans = [PMDOffset 0 PMD_height];
PMDtrans = rigid3d(R,trans);
%% Find and remove empty cells from saved messages
%Find Empty cells
PMDidx = find(cellfun(@isempty,PMDMessages));
poseidx = find(cellfun(@isempty,poseMessages));



%Remove empty cells
PMDMessages(PMDidx) = [];
poseMessages(poseidx) =[];

%% Remove Frames with mismatched Ids
%Extract PMD frame ids and save in numeric vector
PMDFrameIds = [];
for i = 1:length(PMDMessages)
    PMDFrameIds(i) = str2num(PMDMessages{i}.Header.FrameId);
end
%Extract pose frame ids and save in numeric vector
poseFrameIds =[];
for i = 1:length(poseMessages)
    poseFrameIds(i) = str2num(poseMessages{i}.Header.FrameId);
end





%Compare and adjust frame id vectors so they are the same.
poseFrameIds = poseFrameIds(ismember(poseFrameIds,PMDFrameIds));
PMDFrameIds = PMDFrameIds(ismember(PMDFrameIds,poseFrameIds));
```

```
%Remove frames from cell array that do not match
updateLength = length(PMDFrameIds);
for i = 1:updateLength
    localFrameId = str2num(PMDMessages{i}.Header.FrameId);
    if ~any(PMDFrameIds == localFrameId)
        PMDMessages(i) = [];
    end
    updateLength=length(PMDMessages);
end
for i = 1:updateLength
    localFrameId = str2num(poseMessages{i}.Header.FrameId);
    if any(poseFrameIds == localFrameId)
    else
        poseMessages(i) = [];
    end
    updateLength=length(poseMessages);
end

%% Read remaining messages with matching frame Ids
idx = length(PMDMessages);
ptCloud = cell(idx,1);



for k = 1:idx
    %Read PMD messages and rotate to world coordinate frame.
    PMDMsg = PMDMessages{k};
    PMDImage = readImage(PMDMsg);
    PMDCloud = pointCloud(PMDImage(:,:,1:3));
    PMDCloud = pctransform(PMDCloud,PMDtrans);
    ptCloud{k} = PMDCloud;

    %Convert quat Pose to 2D relative Pose
    poseMsg = poseMessages{k};
    Eul = quat2eul([poseMsg.Pose.Orientation.W
poseMsg.Pose.Orientation.X poseMsg.Pose.Orientation.Y
poseMsg.Pose.Orientation.Z]);
    theta = Eul(1);
    wPose2D(k,:) = [poseMsg.Pose.Position.X poseMsg.Pose.Position.Y
theta];

end

%% Compute  initial transforms from 2D data
tic
initTform = rigid3d;
initialTransforms = cell(length(ptCloud),1);
for i = 1: length(ptCloud)
    if i == 1
        trans =[0 0 0];
    else
        trans = [wPose2D(i,1) wPose2D(i,2) 0];
    end
    rot = rotz(-rad2deg(wPose2D(i,3)));
    initTform = rigid3d(rot,trans);
    initialTransforms{i} = initTform;
end


%% Transform point clouds using 2D data

ptCloudInitTform =cell(length(ptCloud),1);
for n = 1: length(ptCloud)
    if n == 1
        ptCloudInitTFrom{n} = ptCloud{n};
    end

   ptCloudInitTform{n} =pctransform(ptCloud{n},initialTransforms{n});
end

%% Find relative transform using icp and add to vSet
relTform= rigid3d;
absTform = rigid3d;

for k = 1:length(ptCloudInitTform)
    if k == 1
        vSet =
addView(vSet,k,initialTransforms{k},"PointCloud",ptCloudInitTform{k});
        absTform = rigid3d(initialTransforms{k}.T);
    else
        prevPC = ptCloudInitTform{k-1};
        [~,~,outliers] = pcfitplane(prevPC,.1,[0 0 1],15);
        prevPC = select(prevPC,outliers,'OutputSize','full');

        currentPC = ptCloudInitTform{k};
        [~,~,outliers] = pcfitplane(currentPC,.1,[0 0 1],15);
        currentPC = select(currentPC,outliers,'OutputSize','full');

        relTform =
pcregistericp(currentPC,prevPC,'Metric','pointToPoint','Extrapolate',tr
ue);
        absTform = rigid3d(absTform.T * relTform.T);
        vSet =
addView(vSet,k,absTform,"PointCloud",ptCloudInitTform{k});
    end
end



%% Build Map and view
gridSize =0.03;
ptCloudMap = helperBuildMapFromViewset(vSet, gridSize);
toc
pcshow(pcdenoise(ptCloudMap));
hold on
scatter3(wPose2D(:,1),wPose2D(:,2),PMD_height*ones(length(wPose2D),1));
hold off
xlabel('x');ylabel('y');zlabel('z');
```

```
%% Optional: Save Data
saveInput = input('Save Data (y/n)?\n','s');
if strcmp(saveInput,'y') || strcmp(saveInput,'Y')
    name = input('Enter File Name:\n','s');
    fileName = sprintf('%s.mat',name);
    save(fileName);
end


%% Supporting Functions


function ptCloudMap = helperBuildMapFromViewset(vSet, gridSize)
%This function is adapted from a function included in MATLAB R2020a
example
%BuildAMapFromLidarDataUsingSLAMExample.m. It utilizes point cloud
viewsets
%instead of the pcmerge.m function included in earlier work.

numViews = vSet.NumViews;

% Extract point cloud views and absolute transformations from view set.
ptClouds  = vSet.Views.PointCloud;
absTforms = vSet.Views.AbsolutePose;

% Make point clouds unorganized
for n = 1 : numViews
    ptClouds(n) = removeInvalidPoints(ptClouds(n));
end

% Preallocate map points
totalNumPoints = sum([ptClouds.Count]);
mapPoints = zeros(totalNumPoints, 3, 'like', ptClouds(1).Location);

pointIndex = 1;
for n = 1 : numViews
    % Transform points to reference frame of first point cloud
   % ptCloud = pctransform(ptClouds(n), absTforms(n));
    ptCloud = ptClouds(n);
    % Accumulate map points
    count = ptCloud.Count;
    mapPoints(pointIndex:pointIndex+count-1, :) = ptCloud.Location;

    pointIndex = pointIndex + count;
end

% Downsample points using voxel grid filter to the requested resolution
ptCloudMap = pcdownsample(pointCloud(mapPoints), 'gridAverage',
gridSize);
end
```

APPENDIX B. ROBOT 1

A. ROBOT_1_LLAM.M

```
%Robot_1_LLAM.m is the full LLAM algorithm written to control Robot 1.
%It was not fully debugged or checked for error handling prior to the
%construction of Robot 2, however it had several succesful trials.
Goals are frontier points selected from CannyFrontiers.m, waypoints are
the intermediate points along the path that are output from the Hybrid
A* Search algorithm.

clear
clc
close all
addpath 'C:\Users\localadmin\OneDrive - Naval Postgraduate
School\NPS\Thesis\MATLAB Code\myMatlab'

%% Navigation Parameters
goal=[4;0 ].*610;
waypointCurrent=goal;
global goalTolerance
goalTolerance =300;
goalReached = false;
numGoalsReached = 0;
numWaypointsReached = 0;
maxGoals = 3;



%% Connect to Robot
p3_connector_Payne('Com8');
disp('Connected to Robot')
pause(0.8)

%% Connect to Hokuyo LIDAR via usb
%Open Connection To Hokoyu

hokoyuAttempt=0;
hokoyuConnected=0;
while hokoyuAttempt<3 && hokoyuConnected == 0
    try
    s='Com7';
    t=serial(s,'InputBufferSize',5000);
    fopen(t);
    fprintf(t,'BM\n'); % measurement state
    pause(0.1)
    data = fread(t,t.BytesAvailable);
    fprintf(t,'II\n'); % information
    pause(0.1)
    data = fread(t,t.BytesAvailable);
    char(data')
    hokoyuConnected == 1;
    catch ME
        disp(['ID: ', ME.identifier])
        warning('Trying to Connect to Hokuyo Again')
        hokoyuAttempt=hokpyuAttempt + 1;
    end
end

disp('Connected to Hokuyo')
pause(0.8)

%Connect to PMD Monstar
%See initialization local function in Master.m
[cameraDevice, camlist, manager,ExposureMode]=Initialize();
cameraDevice.startCapture();
disp('Connected to PMD Pico Monstar')

%Monstar Parameters
sensor_height = 0.435;
world_rotation_matrix=affine3d([0    -1     0   0;
                                 0     0    -1   0;
                                 1     0     0   0;
                                 0     0     sensor_height   1;]);
%Monstar Depth Confidence
depth_confidence=155;

%Hokoyu Parameters
start=0;
stop=1080;
skip=0;
res=0.25;        %URG-30LX-EW
angleVector=((res*start:res:stop*res)-135)'.*pi/180;
minScanRange = 0.3;

 %Pose graph Parameters
pGraph2d=poseGraph;



%Create SLAM Object
MaxFrames = 500;
maxRange = 25; % meters
resolution = 10; % cells per meter
slamObj = lidarSLAM(resolution,maxRange);
slamObj.LoopClosureThreshold = 100;
slamObj.LoopClosureSearchRadius = 3;



%Create Occupancy Map
width = 30;
height = 30;
occMap=occupancyMap(width,height,resolution);
```

```
%Create goal and route validator
global validator
validator=validatorOccupancyMap;

global planner
planner=plannerHybridAStar(validator);






tic
i=0;
scans=cell(1,MaxFrames);
scans3d=cell(1,MaxFrames);
lastAddIndex=1;

while p3_getBumpersClear  & i<MaxFrames & numGoalsReached<=maxGoals
    i=i+1
    %Get 2d Lidar Data
    rangeVector= utmGetScan(t,start,stop);
    scan2d=LidarScan2d_fun(rangeVector,angleVector,minScanRange);
%Create Lidar Scan for mapping
    scans{i}=scan2d; %Save for Map Building

    %Get 3d Lidar Data
    data=cameraDevice.getData();
    depdata=single(data.depthConfidence(:));
    points=[data.x(:) data.y(:) data.z(:) depdata];
    idx = points(:,4)>depth_confidence;
    points((idx==false),:)=[];
    points(:,4)=[];
    lidar_pc=pctransform(pointCloud(points),world_rotation_matrix);
    scans3d{i}=lidar_pc;

    if i >= 2
        if i == 2
            p3_setTransVel(400);
        end
        relPose=matchScans(scans{i-1},scans{i});
        distanceMoved=norm(relPose);
        addRelativePose(pGraph2d,-relPose);
        if distanceMoved>0.03
            poses=nodes(pGraph2d);
            poseCurrent=poses(end,:);
            waypointReached =
p3_goalReached(poseCurrent,waypointCurrent);

            %Initial Goal check, compute map and find next goal
            if waypointReached == true && numGoalsReached == 0
                p3_setTransVel(0);
                p3_setRotVel(0);
                for j=lastAddIndex:10:i
                    addScan(slamObj,scans{j});
                    [scansSLAM,poses] = scansAndPoses(slamObj);
                    occMap =
buildMap(scansSLAM,poses,resolution,maxRange);
                    lastAddIndex = i;

                end
                [refpath,isValid]=CannyFrontiers(occMap,poseCurrent);
                if isValid == false
                    break
                end

                waypoints=refpath.States(1:end,1:2);
                goal=waypoints(end,:);
                numGoalsReached = numGoalsReached + 1;
                numWaypointsReached = 1;
                waypointCurrent=waypoints(numWaypointsReached,1:2);

            %Perform at intermediate waypoints
            elseif waypointReached == true &&
numWaypointsReached<length(waypoints)
                numWaypointsReached = numWaypointsReached + 1;
                waypointCurrent=waypoints(numWaypointsReached,1:2);

            %Perform if last waypoint is the goal
            elseif waypointReached == true
                p3_setTransVel(0);
                p3_setRotVel(0);
                numGoalsReached= numGoalsReached + 1;
                    for j=lastAddIndex:10:i
                    addScan(slamObj,scans{j});
                    [scansSLAM,poses] = scansAndPoses(slamObj);
                    occMap =
buildMap(scansSLAM,poses,resolution,maxRange);
                    lastAddIndex = i;

                    end
                [refpath,isValid] = CannyFrontiers(occMap,poseCurrent);
                if isValid == false
                    break
                end
                waypoints=refpath.States(1:end,1:2);
                goal=waypoints(end,:);
            end

[fwdVel,rotVel]=potentialField(poseCurrent,lidar_pc,scan2d,waypointCurr
ent);
            p3_setTransVel(fwdVel);
            p3_setRotVel(rotVel);
        end
    end
end
 %End while loop
```

```
time=toc;

%%%%Close Hokoyu LIDAR%%%%%%%
utmClose(t);
disp('Hokoyu Disconnected')
%%%%%Disconnect from robot%%%%%%
p3_stopRobot;
p3_disconnector;
%%%%%%Disconnect from Pico Monstar%%%%%%%
cameraDevice.stopCapture();
disp('PMD Pico Monstar Disconnected')
fprintf('Capture Speed = %.3f (Hz)\n',i/time);



%Uncomment to show occupancy map and overlaid 2D trajectory
figure
show(occMap)
title('Occupancy Map of Lab')
hold on
show(pGraph2d)
hold off

%Build 3D map using 2D transformation method
merGridStep = 0.01;
ZregionLimits = [-.2,2.2];
accumTform=affine3d;
tform=affine3d;
k=sum(~cellfun(@isempty,scans3d),2);
tformout=cell(1,k);
transforms=nodes(pGraph2d);
tic

for index=2:k
    if index==2
       moving=pcdownsample(scans3d{index},'gridAverage',0.05);

       tform.T=[cos(transforms(index,3)), -sin(transforms(index,3)), 0,
transforms(index,1);
                sin(transforms(index,3)), cos(transforms(index,3)), 0,
transforms(index,2);
                0                          0                         1
0;
                0                          0                         0
1]';
       aligned=pctransform(moving,tform);
       scene=pcmerge(scans3d{index-1},aligned,merGridStep);

       accumTform=tform;
    elseif index>2

        moving=pcdownsample(scans3d{index},'gridAverage',0.05);
```

```
         tform.T=[cos(transforms(index,3)), -sin(transforms(index,3)),
0, transforms(index,1);
                sin(transforms(index,3)), cos(transforms(index,3)), 0,
transforms(index,2);
                0                           0                         1
0;
                0                           0                         0
1]';
        accumTform=affine3d(tform.T*accumTform.T);
        aligned=pctransform(moving,tform);
        scene=pcmerge(scene,aligned,merGridStep);
    end
end




timeToRender=toc;

%Display 3D map
fprintf('Time to render = %.3f seconds\n',timeToRender);
ind=scene.Location(:,3)>ZregionLimits(1)&scene.Location(:,3)<ZregionLim
its(2);
ptCloudCut=select(scene,ind,'OutputSize','Full');
pcshow(ptCloudCut);
hold on
show(slamObj);
title('Map of Lab')
xlabel('X');
ylabel('Y');
hold off
```

B. CANNYFRONTIERS.M

```
function [refpath,isValid] = CannyFrontiers(occMap,poseCurrent)
%CannyFrontiers.m formats an occupancy map as a binary image. Detects
edges using Canny edge detection. Detects frontiers using a
convolutional filter. Then plans a route using Hybrid A* Search.

global validator
global planner

% Create temporary occ map so we do not inflate the original
tempOccMap=occupancyMap(occMap);
inflate(tempOccMap,0.1)

% %Create validator Object
validator.Map=tempOccMap;
planner.MinTurningRadius=1.3;
```

```
%Create ternary matrix from occupancy map and run Canny edge detection
A = occupancyMatrix(occMap,'ternary');
Aprime =A ==0;
BW = edge(Aprime,'Canny');

%Find Indices of edges
[r,c]=find(BW==1);

%initialize variables for loop
indices=[];
out=[];

%Check edges to see which pixels are unoccupied and adjacent to uknown
%using convolution filter
for ix = 1:length(r)
    if A(r(ix),c(ix)) == 0
    B=zeros(size(A));
    B(r(ix),c(ix))=1;
    out=A(conv2(B,[1 1 1;1,0,1;1,1,1],'same')>0);
    end

    if any(out<0)
        indices(ix,:)=[r(ix),c(ix)];
    end

end
indices=indices(indices~=0);
indices=reshape(indices,[length(indices)/2,2]);

%Create an image of only the frontier edges
frontiers=zeros(size(A));
for i=1:length(indices)
    j=indices(i,1);
    k=indices(i,2);
    frontiers(j,k)=1;
end



%Create world reference object
imSize=size(frontiers);
xWorldlimits=occMap.XWorldLimits;
yWorldlimits=occMap.YWorldLimits;
R=imref2d(imSize,xWorldlimits,yWorldlimits);

frontiers=flipud(frontiers); %Needed for intrinsicToWorld to not flip y
values
cc=bwconncomp(frontiers);   %Find connected components in frontiers
%Get stats on frontier ellipses
stats =
regionprops('table',cc,'Area','Centroid','MajorAxisLength','MinorAxisLe
ngth','Orientation');
```

```
stats=flipud(sortrows(stats));

isValid = false;
frontierArea=1000; %initialize with large value
t = linspace(0,2*pi,48); %Parameter for inscribing ellipse

while isValid == false & frontierArea > 3
    inc = 1;

    %Inscribe ellipse
    a = stats.MajorAxisLength(inc)/2;
    b = stats.MinorAxisLength(inc)/2;
    Xc = stats.Centroid(inc,1);
    Yc = stats.Centroid(inc,2);
    phi = deg2rad(-stats.Orientation(inc));
    x = Xc + a*cos(t)*cos(phi) - b*sin(t)*sin(phi);
    y = Yc + a*cos(t)*sin(phi) + b*sin(t)*cos(phi);
    [xEllipse,yEllipse]=intrinsicToWorld(R,x,y);

%Check validity of points on ellipse with Hybrid A* validator
    validCandidates=[];
    for ix=1:length(xEllipse)
        goalCandidate=[xEllipse(ix) yEllipse(ix)];
        goalCandidateVec=goalCandidate-[poseCurrent(1) poseCurrent(2)];
        thetaCandidate=atan2(goalCandidateVec(2),goalCandidateVec(1));
        goalCandidate=[goalCandidate thetaCandidate];
        candidateValid=isStateValid(validator,goalCandidate);
        if candidateValid == true
            validCandidates=[validCandidates goalCandidate'];
        end
    end
    if isempty(validCandidates)
        break
    end
    %Calculate distances to valid ellipse points
    distances=vecnorm(validCandidates(1:2,:)-[poseCurrent(1)
poseCurrent(2)]');
    goal=validCandidates(:,find(distances==min(distances)))';
    isValid = isStateValid(validator,goal);
    inc = inc + 1;
end

if isValid == false
    warning('No Valid Frontier Found')
    refpath=0;
else
    try
        refpath=plan(planner,poseCurrent,goal);
        figure
        show(planner);

    catch ME
        refpath = 0;
        isValid = false;
        disp(['ID: ', ME.identifier])
        warning('No Valid Path Found')
    end
end





%Optional: plot frontier ellipses on binary frontier image

%Uncomment to show frontier ellipses in a figure
% figure
% imshow(frontiers);
% sz=size(stats);
% t = linspace(0,2*pi,50);
% hold on
% for k = 1:sz(1)
%     a = stats.MajorAxisLength(k)/2;
%     b = stats.MinorAxisLength(k)/2;
%     Xc = stats.Centroid(k,1);
%     Yc = stats.Centroid(k,2);
%     phi = deg2rad(-stats.Orientation(k));
%     x = Xc + a*cos(t)*cos(phi) - b*sin(t)*sin(phi);
%     y = Yc + a*cos(t)*sin(phi) + b*sin(t)*cos(phi);
%     plot(x,y,'r','Linewidth',1)
% end
% hold off

%Optional: Uncomment to plot selected frontier
% figure
% imshow(frontiers)
% hold on
%
% a=stats.MajorAxisLength(maxRegionidx)/2;
% b = stats.MinorAxisLength(maxRegionidx)/2;
% Xc = stats.Centroid(maxRegionidx,1);
% Yc = stats.Centroid(maxRegionidx,2);
% phi = deg2rad(-stats.Orientation(maxRegionidx));
% x = Xc + a*cos(t)*cos(phi) - b*sin(t)*sin(phi);
% y = Yc + a*cos(t)*sin(phi) + b*sin(t)*cos(phi);
% plot(x,y,'r','Linewidth',1)
%
%
% hold off

%Uncomment to show extrema on occMap
% figure
% show(occMap)
% hold on
% plot(xWorldExtrema,yWorldExtrema,'+')
% hold off
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. POTENTIAL FIELD MODEL AND SERIAL COMMUNICATIONS

A. POTENTIALFIELD.M

```
function
[fwdVel,rotVel,Frep_r,Fatt_r]=potentialField(poseCurrent,scan3d,scan2d,
goal,mode)

%potentialField.m calculates the potential field, attractive and
%repulsive forces, and steering commands for the robot. The function
%returns the forward and rotational velocities which are then sent to
p3_setTransVel.m
%and p3_setRotVel.m. This script is adapted from SONAR to 2D and 3D
%LIDAR based on the method in:
%X. Yun and K. Tan, “A Wall-Following Method for Escaping Local Minima
in Potential Field Based Motion Planning,”
%in 1997 8th International Conference on Advanced Robotics.
%Proceedings. ICAR’97, pp. 421–426, Jul. 1997.

%Local variables for input arguments
qGoal=goal.';
q=[poseCurrent(1); poseCurrent(2)].*1000;
theta= poseCurrent(3);

%Potential Field Calculations
RHO = 500;          % use with attractive force
ZETA =  3.2;        % use with attractive force
ETA = 4000;         % constant coefficient
sensX = 1;          % sensitivity for transVel
sensY = 15;         % sensitivity for rotVel
dC = 3000;          % cut-off distance

%Find 3D Points that fall within height of robot and sensors
ind = (0.07<scan3d.Location(:,3)) & (scan3d.Location(:,3)<0.75);
ind=find(ind);
slice3d=select(scan3d,ind,'OutputSize','full');
xyCoords=[slice3d.Location(:,1), slice3d.Location(:,2)]'.*1000;

%Convert from robot coordinates to polar coordinates
lidarRanges=vecnorm(xyCoords);
gamma=atan2(xyCoords(2,:),xyCoords(1,:))'.*1000;
gammaBins=deg2rad([50; 40; 30; 20; 10; 0; -10; -20; -30; -40; -50]);
dev=deg2rad(0.25);
gammaSaved=[];
rangeSaved=[];

%Compress 3D points into 2D Hokuyo plane, save only one point in each
angle
%bin
for i=1:length(gammaBins)
    idx= gammaBins(i)-dev<=gamma & gamma<=gammaBins(i)+dev;
    if ~numel(idx) >= 2
        idx = idx(1:2);
    end
    gammaTemp=gamma(idx)';
    rangeTemp=lidarRanges(idx);
    gammaSaved=horzcat(gammaSaved, gammaTemp);
    rangeSaved=horzcat(rangeSaved, rangeTemp);
end
gamma=gammaSaved;
lidarRanges=rangeSaved;

%Convert m to mm and combine 2D and 3D data
ranges2d=scan2d.Ranges'.*1000;
angles2d=scan2d.Angles';
lidarRanges=[lidarRanges ranges2d];
gamma=[gamma angles2d];




% compute the ATTRACTIVE FORCE in Robot-coords,
if(norm(q-qGoal) <= RHO)
    Fatt_w = -ZETA*(q-qGoal);
else
    Fatt_w = -ZETA*RHO * (q-qGoal)/norm(q-qGoal);
end

% transform Fatt_w to Fatt_r
Tw2r = [cos(theta) sin(theta); -sin(theta) cos(theta)];
Fatt_r = Tw2r * Fatt_w;  % use this later in eq. 7



Frep_r = [0;0];
for ix = 1:length(lidarRanges)
    if(lidarRanges(ix) <= dC)
        Rs2r = [cos(gamma(ix))   -sin(gamma(ix));  % sensor to robot
frame
            sin(gamma(ix))      cos(gamma(ix))];
        ni = Rs2r * [lidarRanges(ix);0]  ;
        di = lidarRanges(ix);
        Frep_r = -ETA * (1/di - 1/dC)*...
            ni./di + Frep_r;

    end
end

%Sum attractive and repulsive forces and compute steering commands
Ftotal_r = Frep_r + Fatt_r;
fwdVel = sensX * Ftotal_r(1);
dir = atan2(Ftotal_r(2),Ftotal_r(1));
rotVel = sensY * dir;

%These parameters are optional and can be adjusted based on the
```

```
%environment. In some cases having the robot turn slowly or not move in
a
%backward direction is beneficial to avoiding a local minimum or
matching
%degeneration. RECOMMENDED: limit forward velocity to less than 500
mm/s.
if dir > pi/2 || dir< -pi/2
    %Limits the robot turning to quickly
    %if the goal is more than 90deg from current heading
    if fwdVel>0
        fwdVel=40;
    elseif fwdVel<0
        fwdVel=-40;
    end

    if rotVel>0
        rotVel=10;
    elseif rotVel<0
        rotVel=-10;
    end
else
    %If the goal within 90deg of current heading, limits maximum
velocities
    if fwdVel > 225
        fwdVel = 225;
    end

    if rotVel > 40
        rotVel = 40;
    end
end
```

B. P3_CONNECTOR.M

```
function p3_connector(comString)
% p3_connector initializes the connection to the robot. This script was
% adapted from one written by Dr. James Calusdian, NPS ECE dept.
% p3_connector(comString) opens the communication with either the real
% robot or MobileSim.  To connect to the actual robot the input
%parameter comString must be set equal to 'Com1' or appropriate com
%port. To connect to MobileSim, comString must be set to 'MobileSim'.
% Also see p3_disconnector for additional information.

% in case we have some ports open from previous failed connections
if ~isempty(instrfindall)
    delete(instrfindall);
end

if ~isempty(timerfindall)
    delete(timerfindall)
end
```

```
global robotConnector;
global SIP_HANDLER;
global PULSE;

% first define the sync bytes that we need to use
SYNC0 = uint8([250 251 3 0 0 0]);
SYNC1 = uint8([250 251 3 1 0 1]);
SYNC2 = uint8([250 251 3 2 0 2]);
START_SERVER = uint8([250 251 3 1 0 1]);
ENABLE_MOTORS = uint8([250 251 6 4 59 1 0 5 59]);


% also define the constants and variables we need
syncState = 0;  % switch parameter
sync0Lock = 0;  % case parameter
sync1Lock = 1;  % case parameter
sync2Lock = 2;  % case parameter
syncLock012 = false; % overall sync status
tryCounter = 0;      % number of attempts to communicate
MAX_TRIES = 3;       % number of times to try synching up with robot



% determine what type of input we have
if nargin==0
   s1 = sprintf('p3_connector FAIL!  Must provide an input parameter');
   s2 = sprintf('Exiting connector function.\n');
   disp(s1);
   disp(s2);
   return;
else
    if strcmp(comString, 'MobileSim')
        s = sprintf('Connecting to MobileSim...');
        disp(s);
        % define our tcip connection
        robotConnector = tcpip('localhost',8101);  % connecto to
MobileSim
        %set(robotConnector,'Terminator','');
        fopen(robotConnector);      % open the connection

    elseif strcmp(comString, comString)
        s = sprintf('Connecting to real robot on Com1...');
        disp(s);
        % establish serial connection to the real robot...
        robotConnector = serial(comString,'BaudRate',9600);
        fopen(robotConnector);

    else
        s1 = sprintf('Input parameter not recognized');
        s2 = sprintf('Exiting p3_connector function.\n');
        disp(s1);
        disp(s2);
        return;
    end
end


% send and verify our syncronization packets
while( ~syncLock012 )

 switch syncState

     case sync0Lock
         s = sprintf('Sending Sync0');
         disp(s);
        fwrite(robotConnector, SYNC0);
        [response, counts] = fread(robotConnector, [1 6],'uint8');
        if isequaln(response, SYNC0)
            syncState = sync1Lock;
            s = sprintf('Sync0 acknowledged\n');
            disp(s);

        else
            if tryCounter < MAX_TRIES
                syncState = sync0Lock;
                syncLock012 = false;
                tryCounter = tryCounter + 1;
                s = sprintf('Sync0 fail.   Sending Sync0 again\n');
                disp(s);
            else
                syncLock012 = true;  % set to TRUE to get out of while-
loop
                s = sprintf('Sync0 fail. Max tries exceeded\nClosing
local port\n');
                disp(s);
                fclose(robotConnector);
            end

        end

     case sync1Lock
         s = sprintf('Sending Sync1');
         disp(s);
         fwrite(robotConnector, SYNC1);
         [response, counts] = fread(robotConnector, [1 6], 'uint8');
         if isequaln(response, SYNC1)
            syncState = sync2Lock;
            s = sprintf('Sync1 acknowledged\n');
            disp(s);
            syncLock012 = false;

         else
            if tryCounter < MAX_TRIES
                syncState = sync1Lock;
                syncLock012 = false;
                tryCounter = tryCounter + 1;
                s = sprintf('Sync1 fail.   Sending Sync1 again\n');
                disp(s);
            else
                syncLock012 = true;  % set to TRUE to get out of while-
loop
                s = sprintf('Sync1 fail. Max tries exceeded\nClosing
local port\n');
                disp(s);
                fclose(robotConnector);
            end
         end

     case sync2Lock
         s = sprintf('Sending Sync2');
         disp(s);
         fwrite(robotConnector, SYNC2);pause(0.8);
         bytesAvail = robotConnector.BytesAvailable;
         [response, counts] = fread(robotConnector, [1
bytesAvail],'uint8');
         s = sprintf('Connected to %s\n',response(3:end-3));
         disp(s);

         % send the OPEN command to start up server
         fwrite(robotConnector, START_SERVER);

         % start up heartbeat timer
         p3_heartbeatTimer;
         answer = PULSE.Running;
         s = sprintf('Heartbeat timer is %s\n',answer);
         disp(s);

         % start the SIP handler (timer) to read packets
         p3_SIP_Timer;
         answer = SIP_HANDLER.Running;
         s = sprintf('SIP Handler is %s\n',answer);
         disp(s);

         % send command 4 to enable the motors
         fwrite(robotConnector, ENABLE_MOTORS); pause(0.8)

         % break out of this loop
         syncLock012 = true;



 end  % switch-case

end  % while


pause(5);   % wait a few seconds for everything to sync up
```

C. P3_DISCONNECTOR.M

```
function p3_disconnector
% P3_DISCONNECTOR disconnects from the robot or MobileSim. This script
% was written by Dr. James Calusdian, NPS ECE Department.
% This function disconnects from the robot by stopping the SIP handler,
% stopping the PULSE timer, and closing the robotConnector object.  No
%input or output parameters are required for this function.  Also see
%p3_connector.

global robotConnector;
global SIP_HANDLER;
global PULSE;
global myTestData;

% first define the sync bytes that we need to use
CLOSE_CONNECTION = uint8([250 251 3 2 0 2]);



% stop processing the SIP packets
s = sprintf('Stopping the SIP handler...');
disp(s);
stop(SIP_HANDLER);
answer = SIP_HANDLER.Running;
s = sprintf('SIP Handler is %s\n',answer);
disp(s);
delete(SIP_HANDLER);
%save('testData.mat','myTestData');

% stop the heartbeat, which was started with p3_heartbeatTimer.
s = sprintf('Stopping PULSE heartbeat...');
disp(s);
stop(PULSE);
answer = PULSE.Running;
s = sprintf('Pulse heartbeat is %s\n',answer);
disp(s);
delete(PULSE);


% close the robot connection
s = sprintf('Closing robot connection');
disp(s);
fwrite(robotConnector,CLOSE_CONNECTION); pause(0.8);
fclose(robotConnector);
delete(robotConnector);
```

D. P3_GETBUMPERSCLEAR.M

```matlab
function [bumpersClear] = p3_getBumpersClear
%P3_GETBUMPERSCLEAR returns true if ALL bumpers clear, false otherwise.
%This script was written by Dr. James Calusdian NPS ECE Department.

global SIPdata;
HEADER_BYTE0 = uint8(250);
HEADER_BYTE1 = uint8(251);

% if we have the SIPdata available, we can pull out the battery
voltage.
% First, let's double check that we have the right data
if SIPdata(1) == HEADER_BYTE0
    if SIPdata(2) == HEADER_BYTE1
        bumperStatus = make16(SIPdata(17),SIPdata(16));

       if bumperStatus
           bumpersClear = false;
           disp('Bumper hit!');
       else
           bumpersClear = true;
           %disp('TRUE');
       end
    end
end
```

E. P3_SETROTVEL.M

```matlab
function p3_setRotVel(rotVel)
%P3_SETROTVEL causes the robot to rotate ccw(+) or cw(-) at specified
%deg/sec.p3_setRotVel(rotVel) causes the robot to rotate with the
%angular velocity specified in rotVel (deg/second).  A (+) rotVel
%produces a CCW rotation, and a (-) rotVel produces a CW rotation, as
%viewed from the top of the robot.
%This function was written by Dr. James Calusdian, NPS ECE Department.
%See ARCOS command 9 in Operations Manual, pp 31.

global robotConnector;
HEADER_BYTE0 = uint8(250);
HEADER_BYTE1 = uint8(251);
commandNumber = uint8(9);

% construct the command to rotate
if rotVel < 0
    argType = uint8(27);    % negative
else
    argType = uint8(59);   % positive
end
```

```
% convert "rotVel" into two bytes of uint8
temp = uint16(abs(rotVel));
[MSB, LSB]=split16(temp);

% next construct the command packet
command = uint8([commandNumber argType LSB MSB ]);
byteCount = uint8(length(command) + 2);  % include 2 checkSum bytes
[chkMSB, chkLSB] = checksum4p3(command);
setRotCommand = uint8([HEADER_BYTE0 HEADER_BYTE1 byteCount command
chkMSB chkLSB]);

% send everything to the robot
fwrite(robotConnector, setRotCommand);
```

F. P3_SETTRANSVEL.M

```
function p3_seTransVel(transVel)
%P3_SETTRANSVEL sets the translational velocity for the robot.
%p3_setTransVel(transVel) causes the robot to move forward (+) or
%backward (-) at the speed of "transVel" mm/sec.
%This function was written by Dr. James Calusdian, NPS ECE Department.
%See ARCOS command 11 in Operations Manual, pp 31.

global robotConnector;
HEADER_BYTE0 = uint8(250);
HEADER_BYTE1 = uint8(251);
commandNumber = uint8(11);

% construct the command to translate
if transVel < 0
    argType = uint8(27);    % negative
else
    argType = uint8(59);   % positive
end

% convert "transVel" into two bytes of uint8
temp = uint16(abs(transVel));
[MSB, LSB]=split16(temp);

% next construct the command packet
command = uint8([commandNumber argType LSB MSB ]);
byteCount = uint8(length(command) + 2);  % include 2 checkSum bytes
[chkMSB, chkLSB] = checksum4p3(command);
translateCommand = uint8([HEADER_BYTE0 HEADER_BYTE1 byteCount command
chkMSB chkLSB]);

% send everything to the robot
fwrite(robotConnector, translateCommand);
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Velodyne Lidar, “Alpha Prime,” Alpha Prime Lidar Data Sheet, 2019.

[2] ReconRobotics, “Legacy Products,” [Online]. Available: https://reconrobotics.com/products/legacy-products/ [Accessed: May 20, 2020].

[3] C. S. Hargadine, “Mobile robot navigation and obstacle avoidance in unstructured outdoor environments,” M.S. thesis, Dept. Elect. and Comp. Eng., NPS, Monterey, CA, USA, 2017. [Online]. Available: http://hdl.handle.net/10945/56937

[4] A. S. Miyakawa, “Autonomous ground vehicle low-profile obstacle avoidance using 2D LIDAR,” M.S. thesis, Dept. Elect. And Comp. Eng., NPS, Monterey, CA, USA, 2019.[Online]. Available: http://hdl.handle.net/10945/63486

[5] C. Lebrun, “Vision-based terrain classification and learning to improve autonomous ground vehicle navigation in outdoor environments,” M.S. thesis, Dept. Elect. and Comp. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: http://hdl.handle.net/10945/63474

[6] A. Magee, “Place-based navigation for autonomous vehicles with deep learning neural networks,” M.S. thesis, Dept. Elect. and Comp. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: http://hdl.handle.net/10945/64012

[7] F. Moosmann and C. Stiller, “Velodyne SLAM,” in 2011 IEEE Intelligent Vehicles Symposium (IV), Jun. 2011, pp. 393–398.

[8] J. Zhang and S. Singh, “LOAM: Lidar Odometry and Mapping in Real-time,” presented at the Robotics: Science and Systems 2014, Jul. 2014.

[9] X. Ji, L. Zuo, C. Zhang, and Y. Liu, “LLOAM: LiDAR odometry and mapping with loop-closure detection based correction,” in 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Aug. 2019, pp. 2475–2480.

[10] T. Shan and B. Englot, “LeGO-LOAM: Lightweight and ground-optimized LiDAR odometry and Mapping on Variable Terrain,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2018, pp. 4758–4765.

[11] J. Lin and F. Zhang, “Loam_livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV,” Sep. 2019, Accessed: Oct. 08, 2019. [Online]. Available: http://arxiv.org/abs/1909.06700.

[12] MobileRobots Inc., *Pioneer 3 Operations Manual*, MobileRobots Inc, 2008.

[13] Adept Technology Inc., “Pioneer 3-DX,” Pioneer 3-DX Data Sheet, 2011.

[14] PMD Technologies AG., “High-End Development Kit CamBoard pico monster,” CamBoard pico monster Data Sheet, 2018.

[15] Hokuyo, “UTM-30LX.” Accessed May 20, 2020. [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=169

[16] S. Fotiadis, “Hokuyo UTM-30-LX-EW for MATLAB,” MATLAB Central File Exchange. Available: https://www.mathworks.com/matlabcentral/fileexchange/37613-hokuyo-utm-30-lx-ew-for-matlab, Retrieved May 20, 2020.

[17] MathWorks, “What is MATLAB,” Accessed: May 20, 2020. [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html

[18] PMD Technologies AG., “picofamily,” 2020. Accessed May 20, 2020 [Online]. Available: https://pmdtec.com/picofamily/

[19] PMD Technologies AG., *CamBoard pico monstar Getting Started*, PMD Technologies AG, 2017.

[20] C. Debeunne and D. Vivet, “A review of visual-lidar fusion based simultaneous localization and mapping,” *Sensors*, vol. 20, no. 7, p. 2068, Apr. 2020.

[21] J. A. Bærentzen, J. Gravesen, F. Anton, and H. Aanæs, *Guide to Computational Geometry Processing*. London: Springer London, 2012.

[22] P. Corke, *Robotics, Vision and Control*. Berlin Heidelberg: Springer-Verlag, 2013.

[23] S. Saha and S. K. Shukla, *Advanced Data Structures: Theory and Applications*. Boca Raton: Taylor & Francis, 2019.

[24] P. Ullrich and C. Zarzycki, “TempestExtremes: A framework for scale-insensitive pointwise feature tracking on unstructured grids,” *In Geoscientific Model Development*, vol 10, no. 3, Mar., pp. 1069–1090, 2007.

[25] P. Biber and W. Strasser, “The normal distributions transform: a new approach to laser scan matching,” in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), Oct. 2003, vol. 3, pp. 2743–2748.

[26] M. Magnusson, *The Three-Dimensional Normal-Distributions Transform an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. Örebro: Örebro universitet, 2009.

[27] MathWorks, “Point Cloud Registration Overview,” Accessed: May 20, 2020. [Online]. Available: https://www.mathworks.com/help/vision/ug/point-cloud-registration-workflow.html

[28] A. Myronenko and X. Song, “Point Set Registration: Coherent Point Drift,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2262–2275, Dec. 2010.

[29] A. Geiger, P. Lenz, and R Urtasun, “Are we ready for autonomous driving? The kitti vision benchmark suite,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3354–3361.

[30] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, “Towards 3D point cloud based object maps for household environments,” *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, Nov. 2008.

[31] ROS, “About ROS.” Accessed July 15, 2020. [Online]. Available: http://www.ros.org/about-ros/

[32] CappuccinoPC, “SlimPRO SP675P Mini PC.” Accessed 20 July, 2020. [Online]. Available: https://www.cappuccinopc.com/slimpro-sp675p.asp

[33] X. Yun and K. Tan, “A wall-following method for escaping local minima in potential field based motion planning,” in 1997 8th International Conference on Advanced Robotics. Proceedings. ICAR’97, pp. 421–426, Jul. 1997.

[34] P. Jenko, T. Emter, C. W. Frey, T. Kopfstedt, and A. Beutel. “Applications of hybrid a* to an autonomous mobile robot for path planning in unstructured outdoor environments.” in ROBOTIK 2012: 7th German Conference on Robotics. Pp 1–6. 2012.

[35] MathWorks, “plannerHybridAStar,” Accessed: August 6, 2020. [Online]. Available: https://www.mathworks.com/help/nav/ref/plannerhybridastar.html

[36] MathWorks, “Accelerating MATLAB Algorithms and Applications,” Accessed: August 28, 2020. [Online]. Available: https://www.mathworks.com/company/newsletters/articles/accelerating-matlab-algorithms-and-applications.html?CCode

[37] R. B. Rusu and S. Cousins, “3D is here: point cloud library (PCL),” in 2011 IEEE International Conference on Robotics and Automation, May 2011, pp. 1–4.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California