Real-Time Map Manipulation for Mobile Robot Navigation

by

Carlos Ezequiel

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Luther Palmer III, Ph.D.
Yu Sun, Ph.D.
Sudeep Sarkar, Ph.D.

Date of Approval:
March 25, 2013

Keywords: mapping, obstacle-avoidance, path-planning, user-interface, teleoperation

DEDICATION

For Mom and Dad.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis adviser, Dr. Luther Palmer for his patience, guidance and support of my research. I would also like to thank the committee members, Dr. Yu Sun and Dr. Sudeep Sarkar, for giving supportive feedback on how to improve this work.

I would like to thank the following people who contributed to this thesis: Mayur Palankar, for helping me format this thesis manuscript in Latex and for assisting me with my various printing needs, Daniel Standish, for giving me a quick tutorial on the PRM algorithm and Matlab and his input on video streaming options for the robot, and for providing me with some sample code, and Robert Skinner, for helping me conduct user perception tests on multiple participants.

I would like to acknowledge the Fulbright program for enabling me to study in the United States through a graduate scholarship.

Lastly, I would like to thank my father for his continued support of my education.

AMDG†

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Mobile robots are gaining increased autonomy due to advances in sensor and computing technology. In their current form however, robots still lack algorithms for rapid perception of objects in a cluttered environment and can benefit from the assistance of a human operator. Further, fully autonomous systems will continue to be computationally expensive and costly for quite some time. Humans can visually assess objects and determine whether a certain path is traversable, but need not be involved in the low-level steering around any detected obstacles as is necessary in remote-controlled systems. If only used for rapid perception tasks, the operator could potentially assist several mobile robots performing various tasks such as exploration, surveillance, industrial work and search and rescue operations. There is a need to develop better human-robot interaction paradigms that would allow the human operator to effectively control and manage one or more mobile robots.

This paper proposes a method of enhancing user effectiveness in controlling multiple mobile robots through real-time map manipulation. An interface is created that would allow a human operator to add virtual obstacles to the map that represents areas that the robot should avoid. A video camera is connected to the robot that would allow a human user to view the robot's environment. The combination of real-time map editing and live video streaming enables the robot to take advantage of human vision, which is still more effective at general object identification than current computer vision technology. Experimental

results show that the robot is able to plan a faster path around an obstacle when the user marks the obstacle on the map, as opposed to allowing the robot to navigate on its own around an unmapped obstacle. Tests conducted on multiple users suggest that the accuracy in placing obstacles on the map decreases with increasing distance of the viewing apparatus from the obstacle. Despite this, the user can take advantage of landmarks found in the video and in the map in order to determine an obstacle's position on the map.

CHAPTER 1: INTRODUCTION

Autonomous mobile robot navigation through complex and dynamic environments is an ongoing challenge. Although advances in sensor technology, computing power and path planning algorithms have allowed mobile robots to perform at a high level of autonomy in navigating through cluttered space, reliability and cost-effectiveness is still a major concern. Human perception is still more advanced than machine vision in performing general object recognition tasks. In addition, increasing the number of sensors on a robotic system greatly increases its development and maintenance costs. Path planning also becomes more computationally expensive. Thus, there is a need to have a human-in-the-loop of a robotic system in order to augment the robot's perception and judgement.

1.1 Problem

The manner in which a human operator interacts with a robotic system in order to improve the latter's navigation is an enduring problem that has a number of different solutions and implications. The main question is "How can one best capitalize a human operator's perception and path planning skills to improve robot locomotion in a complex environment?" This work investigates how one can best use a human operator in a human-machine collaborative framework.

1.2 Research Objectives

The goal of the project is to investigate the degree of effectiveness of having a human-in-the-loop to aid in robot movement as opposed to allowing the robot to navigate to a designated goal on its own. Our primary measure of effectiveness would be the time it takes for the robot to reach the goal from a certain location. However, the following factors were also considered:

- Usability of the map-based navigation interface
- User accuracy in identifying obstacles and placing forbidden regions in the correct positions on the map

1.3 Research Approach

In order to address the robot navigation problem, a map-based robot navigation interface (RNI) was created, which would allow the human operator to manipulate the robot's map in real-time. The operator will be able to assist the robot in identifying obstacles that the latter cannot perceive through its sensors. The operator could to relay this information to the robot by adding virtual obstacles to represent the unmapped obstacles on the map, and could then send this map easily using the RNI software. A video camera was used so that the operator can see the forward view of the robot. A third-party application was used for live video streaming to the operator's control software.

The advantage of this approach is that the robot can make use of a human operator's perception, which is faster and more accurate at identifying obstacles in a variety of shapes and sizes than its on-board sensors. The interface would enable the operator to give the robot a few "hints" on the best path to go, without directly changing the path. Since the

robot can navigate autonomously, it frees the human operator to perform other tasks, such as monitoring other robots. The system would enable one human operator to potentially control and change the paths of multiple robots simultaneously.

1.4 Applications

The method can be used to develop systems that would allow a group of robots to navigate in a cluttered environment, such as a warehouse, or to perform multiple search and rescue operations simultaneously. An operator looking through the video feed of one robot can identify a hazardous area, such as an oil spill, toxic waste spill or hole in the ground, and mark an obstacle on the map to indicate to the other robots that the area is not traversible. Another application would be for real-time mapping of an unknown environment that would allow for human operator to identify obstacles or areas of interest and mark them on the map in real time.

1.5 Thesis Organization

Chapter 2 discusses related work done to address the problem of mobile robot navigation. Chapter 3 discusses path planning algorithms that were considered in the robot system used. Chapter 4 describes the robot system's hardware and software architecture. Chapter 5 explains how the test was setup in order to determine the robot's navigation with map manipulation, as well as the user's capabilities for identifying obstacles at a distance. Chapter 6 summarizes the test results. Finally, the conclusion and recommendations for future work are given in Chapter 7.

CHAPTER 2: LITERATURE REVIEW

Much work has been done on improving autonomous navigation by mobile robots [1]. There are three general approaches to mobile robot navigation: (1) highly autonomous, (2) teleoperated and (3) semi-autonomous.

The highly autonomous approach focuses on developing systems that allow the robot to navigate in an environment with very little or no human control. This approach seeks to reduce the need to have a skilled robot operator controlling the robot. Such systems, however, would often need multiple sensors and advanced algorithms, which may be both financially and computationally expensive. Reliability is also a concern, as the robot may malfunction and cause damage without a human operator to take emergency responsive action.

The teleoperated, or "drone-type," approach seeks to enhance a human operator's ability to fully control a mobile robot . This approach is currently used in military-type unmanned aerial vehicles (UAV) [2]. The advantage of this approach is that the system can be relatively inexpensive, with respect to sensors needed, and computationally inexpensive since path planning is done by the operator. The disadvantage is that it may require a high degree of human concentration and skill to control the robot effectively, especially for long-duration missions.

The semi-autonomous approach provides a compromise between the above two approaches. The robot would be able to navigate on its own, but a human operator is present to monitor, command or directly control the robot as needed.

In practice, most robot systems are semi-autonomous in nature, due to the need to have a human operator to ensure safe and effective operation [3]. One can think of the above approaches as having different levels of automation (LOA), with the highest being fully autonomous, and the being lowest manually controlled [4]. However, in order to develop a better understanding of the advantages and disadvantages of various levels, the various LOA are summarized into the three approaches described previously. Despite the necessity to have various LOA in order to effectively solve different problems, the current trend for mobile robot systems is to have higher LOA.

The following sections describe recent studies made in each of the three approaches to autonomous navigation, and how they relate to the method proposed in this study.

2.1 Highly Autonomous Approach

In this category, the emphasis is on how to develop highly autonomous navigation algorithms using a combination of different sensors.

Kala et al. [5] developed a navigation system using three different algorithms: genetic algorithm, artificial neural network and A*. The algorithms were run independently of each other. Their results show that using each of the three algorithms, the robot was able to navigate in an environment without collding with any moving obstacles. Also, their results showed that A* algorithm performed the best out of the three. Testing was done using simulation only.

The A* algorithm was considered in this work, due to its speed and simplicity compared to other navigation algorithms that were also considered. More detailed discussion can be found in Chapter 3.

Garrido et al. [6] proposed a sensor-based global path planning algorithm, which combined the global path planning that generates the optimal path towards a goal, with local planning for collision avoidance, in one step. The algorithm is based on the Voronoi Fast Marching (VFM) method. The algorithm computes both global and local path planning every step that the robot takes towards the goal. Although the computations are performed every step, it is relatively efficient, having a complexity O(n), where n is the number of cells in the grid that is generated from the map of the robot's environment.

Testing was performed in simulated environment having an area of $2000m^2$ (medium size). Performance was measured by taking the worst computation time of the algorithm during the simulation run. The worst time was found to be 150ms, which was considered acceptable for most applications. Despite the relative efficiency of the algorithm, its main limitation is still its computational complexity, especially for large environments, as well as the computational differences between the initial (near start) and final (near goal) of task execution.

The proposed work made use of a combined global and local path planning algorithm, as described in [6]. The algorithm used is explained in more detail in Chapter 4.

Cho et al. [7] explore the use of a laser range finder (LRF) for map-based robot navigation and localization in indoor environments. The study focuses on localization by comparing data from a predefined global map and data from the LRF. The global map is represented by features of vertices and patterns of vertices. The robot localizes by matching the vertices in the global map with those computed from LRF data. The technique can be applied to localize multiple robots with a common map.

Although use of a LRF improves the sensing capabilities of a mobile robot and aids in mapping, localization and navigation, the cost of the device may be prohibitively expensive, especially when building systems consisting of multiple heterogenous mobile robots, such as in a manufacturing environment. The proposed work does not make use of a LRF for navigation due to cost considerations, but instead relies on sonar sensors and human vision. However, LRF integration is possible and would improve the system.

A method of localization and navigation using a monocular vision system was proposed by Ehtemam-Haghighi [8]. Self-localization is performed using a vision system, while path planning and obstacle avoidance by fusing data from both the vision system and a range of ultrasonic sensors. Localization is done using the vision system, which detects a pair of red and green lines on the floor known as landmarks. Obstacle avoidance is done using the Vector Field Histogram (VFH) method, which permits detection of unknown objects. Hough transform was used for edge detection.

The work shows that the robot was able to navigate through obstacles towards a given goal, and that the vision system was able to detect and avoid obstacles. However, the system was tested using opaque rectangular-shaped obstacles that had well-defined edges. Obstacles such as a fence or chair, which a human person can identify easily, may be more difficult to detect, and may require more advanced algorithms that are computationally expensive.

Another vision-based navigation paradigm is explored by Santosh et al. [9]. Here, they developed an image-based exploration algorithm, where the robot is able to explore and map an unknown frontier of its environment autonomously using images. Experimental results conducted in an unmodified laboratory corridor setting demonstrate the validity of the approach. This work focuses on mapping an unknown environment rather than navigation. It may possible to extend the method such that the system would be able to

detect an unmapped obstacle in a known environment. This would require that the vision system detect obstacles from a far distance. The work, however, indicates that the camera was tilted downwards by 30 degrees and has a field of view (FOV) of about 50 degrees, which limits the range at which it detects objects.

Correa et al. [10] examines the use of a Kinect sensor as low-cost vision system for mobile robot navigation. Navigation is performed both reactively, using Kinect depth images for distance estimation for collision avoidance, and deliberatively using artificial neural network (ANN) to identify key points (i.e. left turns, right turns, intersections) in Kinect images. The topological map represents hallways as edges and locations as vertices. Preliminary experiments reveal that the system was able to provide safe navigation for the robot in dark (no light) and light conditions. Also, the ANN subsystem, which after training had 316 neurons in the hidden layer and 8 neurons in the output layer, provided very good results on classifying features (92%).

The work demonstrates the capabilities of the Kinect for mobile robot navigation. However, more tests would be needed in complex environments to verify the practicality of the system. The Kinect sensor has a limited range and may not be able to pick up obstacles past a certain distance. However, it provides a low-cost way of sensing obstacles in the environment.

Tsalatsanis [11] studied the control of multi-sensor, multi-robot teams for industrial applications. The system is composed of a set of functional components perform navigation and obstacle avoidance using vision-system and sonar sensors, localization using Kalman filtering to fuse sensor data and fuzzy logic to estimate robot pose, target tracking based on vision system and laser range finder (LRF) for intruder detection and mission planning using limited look-ahead control methodology based on finite automata theory. Experiments were done in both real indoor and outdoor environments to measure the performance of

each functional component. The mission planning component, however, has only been validated in the simulated setting.

Urmson et al. [12] examine the development and performance of Boss, the autonomous vehicle that had won the 2007 DARPA Urban Challenge. The authors summarized several lessons learned by the team that developed Boss. First, off-the-shelf sensors were not sufficient to support autonmous driving in an urban environment. Also, no single sensor alone is capable of meeting the range and coverage demands of urban navigation. Second, mimicking human driving in an urban environment requires a rich representation of roads, curbs, and other obstacles. It was found that Boss waited equally long behind a stopped car and a barrel, while trying to differentiate between the two. A richer representation, however, would require better sensing capabilities and more complex object recognition algorithms.

Third, there needs to be a way to verify and validate the performance of an urban autonomous driving system. Although there are verification and validation strategies for navigation algorithms and other subsystems that do not interact directly with the outside world, a verification method that can measurably determine how well an autonomous system interacts the real world is still an unsolved problem.

Fourth, a sliding scale autonomy would help reduce autonomous vehicle complexity. The study found that in cases where the vehicle would fail and initiate a recovery procedure, it would be simpler for the vehicle to ask assistance from a human operator. In this way, system complexity would be reduced and recovery time would increase significantly. The challenge, however, is to ensure that the vehicle would be sufficiently capable of acting autonomously and that it should request assistance rarely so as not to defeat the purpose of its autonomy.

Lastly, the study found that driving is a social activity and that there are many nuances to human driver interaction that the autonomous vehicle would not be able to detect in its current form, such as right of way, and when the normal rules for traffic need to be violated to ensure smooth flow.

Doitsidis, Murphy, Long, et. al. [13–15] studied the use of a distributed framework for autonomous multi-robot control. The distributed field robot architecture, also known as DFRA or distributed-SFX, builds a layer on top of the existing software architecture used for individual robots. The layer itself is designed around Sun Microsystem's Jini, which is a network architecture for building modular service-based distributed systems. The entire architecture is based on Java, and uses Jini as a middle-ware layer. Work done in [13, 14] describes an integration of distributed-SFX with the MATLAB environment for rapid prototyping of behavioral and control modules. Multi-sensor fuzzy logic controllers, implemented in MATLAB, would evaluate the sensor data and determine the necessary inputs to the robot drive system for navigation. Experimental results show the applicability of the system for controlling two wheeled robots in an outdoor environment.

The advantage of the approach is that the system can be stand-alone or may be part of a multi-layer distributed system where robot control and processing of sensor data are done at various levels. Also, integration of the MATLAB environment allows the user to quickly develop and test navigation algorithms. The disadvantage is that the system is complex due to the various software layers and may require high processing capabilities. The system is ideal as a testbed for developing robot systems, but may be difficult to use in actual production systems.

2.2 Teleoperated Approach

Current research in teleoperation focuses on developing better user interfaces and algorithms for controlling mobile robots, and evaluating the effectiveness of such interfaces. The degree of effectiveness of a user interface is correlated with the level of situational awareness that it affords the user [4]. It is expected that future robotic systems will incorporate different levels of automation, which includes manual teleoperation. Being able to have direct control over a robotic system is still beneficial in certain situations, such as in urban search and rescue operations [16].

2.2.1 Haptic Feedback

Aside from presenting visual information, haptic information appears to improve a user's ability to teleoperate a robot, particularly using a joystick.

Cho et al. [17] describe the application of "force-feedback" mechanism for enhancing user response in teleoperating a mobile robot. The cited work found that using a joystick with haptick feedback, the operator can drive the mobile robot to the goal position much faster and more safely.

A similar study made by Martinez-Palafox et al. [18] describe bilateral operation of mobile robot over a communication channel with constant delays. The mobile robot is controlled with a joystick that has both linear velocity and heading angle control. It was found that humans can safely teleoperate the robot with force reflection.

Another study conducted by Mitsou et al. [19] describe a visual-haptic interface for teleoperation of a mobile robot. The system is to be used for exploration tasks. The proposed interface was found to improve navigation time and operator perception in teleoperating of a mobile robot in exploring polygonal environments.

Incorporating force feedback or reflection is beneficial for robot systems that are mainly teleoperated in nature. However, in semi-autonomous systems, where direct user control is seldom needed and the environment if relatively flat, haptic-feedback may not be necessary.

2.2.2 Augmented Reality

Augmented reality is one way to enhance situational awareness. The idea is to combine information from various sources and present it in a way that reduces cognitive overload.

Nielsen et al. [20] developed an ecological interface combining video, map and robot pose info into a 3D augmented reality display for mobile robot teleoperation. Participants in the study preferred the 3D interface over conventional 2D interfaces. Results show that the operator would navigate the robot further away from obstacles and reach the goal faster using the 3D interface. Three principles on effective user-interface design were extracted from the study: (1) present a common reference frame, (2) provide visual support for the correlation of action and response and (3) allow an adjustable perspective. The above principles aided in reducing the cognitive processing required to interpret info from the robot and make decisions. Cognitive processing (workload) was measured using NASA-TLX.

Unlike in [20], the system proposed in this paper does not make use of a common reference frame, where map, video and robot pose information are combined in a 3D representation of the environment. Instead, two reference frames are displayed, where one shows a map that displays the robot in its environment, and the other is a video that shows the robot's point-of-view. Although the cited work claims that this would lead to increased cognitive processing, the user is not required to teleoperate the robot, and simply has to

look out for obstacles blocking the path of the robot, which may induce less cognitive load than if the person were to directly control the robot. This 2D map and video approach is simpler to implement than the 3D approach proposed in [20].

The system proposed in this paper provides visual support for correlating action and response. An adjustable, perspective in the form if pan and zoom controls is also provided in the map-based interface. The camera view, however, is fixed.

Kelly et al. [21] developed a method of visualization called "virtualized reality" to allow a synthetic line-of-sight view of the robot. As opposed to the 3D view proposed in [20], which embeds the camera view of the robot in the 3D map, the view proposed in this work combines images from different perspectives of the robot and forms a realistic 3D image of the robot's environment. Tests conducted with different users support the effectiveness of this system. The robot vehicle used is a retrofitted LandTamer that has been equipped with multiple video cameras and LIDAR.

2.2.3 Interface Evaluation

Since mobile robot navigation is highly dependent on the user, the effectiveness of the user interface is of prime importance. As such, studies have been made on how to present information to the user and control a mobile robot more effectively.

Shiroma et al. [16] studied the effect of different camera views on teleoperation of the mobile robot. Results show that the view that allows the user to teleoperate the robot with high efficiency is when the robot at the center and its surroundings are visible. In addition, the fish-eye camera and omnidirectional camera allowed the user to have better control of the robot compared to the traditional camera view since they enabled the user to visualize more of the robot's surroundings. Performance was measured by the time it took the user to complete a robot task.

Although the fish-eye and omnidirectional cameras provide a greater field-of-view, they are not as common as the regular webcams or video cameras. Also, omnidirectional cameras can be expensive. The top-down view proposed in this study allows the user to see the robot as well as its immediate surroundings. however, it also increases the robot's vertical clearance, which may inhibit movement in certain tight spaces.

Sanders et al. [22] studied human operator performance in controlling a mobile robot when time delays were incorporated. Results show that the users can operate well in simple environments without aid of robot sensors and without time delays. However, in more complex environments with time delays, sensors help to achieve better control.

Gomer et al. [23] studied the relationship of spatial perception of human operators with performance in teleoperating a mobile robot. Tests were conducted in order to check performance of an operator teleoperating the robot and using direct line-of-sight for control. It was found that those who had high spatial ability performed better. User performance was measured using NASA-TLX.

Upham Ellis [24] studied user perception and displays in teleoperating a mobile robot. It was believed that overhead camera placement would lead to higher cognitive load, since the user would have to hold one more representation in their minds other than the actual point-of-view of the robot. However, measurements using NASA-TLX did not indicate added workload from camera placement. Based on a separate survey, participants did experience a higher frustration level in performing the robot tasks when using the overhead camera view, as opposed to the attached camera view. In addition, the study found that a larger screen size was preferred when driving precision was needed.

Although the study claims that the overhead, map-type view would lead to greater frustration on the part of the user, this is only relevant to the case where the user has to directly control the robot. The user must do a mental translation of the robot's pose in

order to correctly decide on the type of input to send to the latter. However, if the user is simply supervising the robot, which can move autonomously most of the time, the mental translation need not happen very often.

2.3 Semi-Autonomous Approach

Similar to teleoperated systems, research in semi-autonomous systems focuses on developing better human-robot interfaces, particularly for enabling single-user, multi-robot control. In order to reduce operator load in the human-robot collaboration however, work is being made in increasing the level of autonomy (LOA) of mobile robots. Since the goal is for the user to take on a supervisory, as opposed to direct, role in mobile robot navigation, new ways of communicating with the robot have been proposed. The common theme of human-robot interaction, aside from increase LOA, is to also increase situational awareness (SA), which indicates how well the user can comprehend the objects perceived from the robot's environment [25].

2.3.1 Single-Robot Interfaces

Doroodgar et al. [25] designed a control architecture for a search-and-rescue mobile robot platform with novel 3D mapping sensor that uses SLAM. The intent is to increase situational awareness by providing 2D and 3D images in real-time to the user. The system makes use of a hierarchical reinforcement learning (HRL) algorithm in order to learn and make decisions to which tasks can be performed safely and autonomously by the robot or would require direct user control. Experimental results show that the semi-autonomous, shared-control approach resulted in less obstacle collisions compared to a fully teleoperated

approach. Also, participants indicate that they experienced more stress in full teleoperation of the robot then in semi-autonomous control.

Koch et al. [26] developed a Java-based, universal web interface for robot control. The work allows the robot to be teleoperated using any Java-enabled browser, making the system platform independent. The intention is to allow the users to have a common graphical interface for controlling a robot through the Internet. Most robot interfaces are tied to the robot hardware, which leads to various interface schemes for different types of robots. This means that the user would have to learn how to use different interfaces for different robot platforms. Having a common interface would reduce the learning curve for controlling new robot platforms, and enabling internet-based control would allow an expert user from a remote location to take command of the robot.

Kadavasal Sivaraman et al. [27] developed a virtual reality (VR)-based multi-modal interface for mobile robot teleoperation. The system uses a VR model and augments it with live-video feed from a stereo-vision camera as well as prediction states. The system can switch between teleoperated and autonomous modes. Experimental results show that the VR-based mixed autonomy system is more effective compared to traditional teleoperation interfaces in partially known environments. The disadvantage is that VR can only be used in environments where the user has prior knowledge about the terrain.

2.3.2 Multi-Robot Interfaces

Nebot et al. [28] developed an interface that enables any user to teleoperate a team of robots to achieve specific goals. The control model allows for both direct control and supervisory control. The interface model is designed for a multi-modal, multi-sensor scheme such that all the information about the robot can be integrated into a unique display. It consists of two types of windows: a main window for displaying a map, and dynamic

robot windows, that change depending on the configuration of the robot (whether it has an LRF, video camera, etc.).

Tang et al. [29] developed a multi-level semi-autonomous mobile robot architecture (SAMRA). The interface has live video/audio, a 3D drawing simulation and a graphic mission planner. Experimental results show that the mission planner is able to perform the set tasks successfully and that the system has excellent telepresence, flexibility and robustness.

Chadwick et al. [30] studied a single-user, multi-robot control interface where each robot's goal is to search for radioactive targets. Experiments on the system was done in simulation. The robot is equipped with a GPS unit, fixed video camera and Geiger counter for radiation sensing. The interface concept has four miniature video displays, one main video display, a map view and control module. Control of each UGV is done through switching, where the main video display shows the robot that would be controlled. It was found that monitoring four different video streams made it difficult for the user to pay attention to each one long enough to understand the action that was depicted. The delay in detecting robot errors went from within 10 seconds for the 1-robot case, to almost 120 seconds in the 4-robot case.

Sato et al. [31] designed a multi-robot control interface that allows formation control by enabling the user to draw a "bounding box" on the map in order to group multiple robots. Motion commands can then be sent to the group. Proposed method currently only allows one group to be defined. The system was implemented on a touch-screen and a touch pen was used for trajectory input. Results show that proposed solution is able to reduce operator load for controlling multiple robots.

Lee et al. [3] designed an augmented reality (AR) interface for control of multiple mobile robots in hazardous environments. The study compared the AR interface to tradi-

tional joystick control and virtual display interfaces. The system made use of an overhead camera to visual multiple robots in one sceen.

2.3.3 Interface Evaluation

Keyes et al. [32] analyzed the degree of situational awareness provided by various human-robot interfaces using location, activities, surroundings, status and overall mission (LASSO) metrics. The authors found that a map-centric interface is more effective at providing good location and status awareness, but a video-centric interface is more effective in providing good surroundings and activities awareness.

Wang et al. [33] studied mixed initiative human-robot interaction on improving the performance of mobile robots. The study suggests that mixed initiative interaction led to better performance than either teleoperation or full autonomy, the reason being that because it is difficult to determine the most effective task allocation a priori, enabling user adjustment during execution should improve performance. Tests were conducted using USARsim, a high-fidelity game engine-based simulator used in testing human-robot interaction for urban search and rescue applications.

Cross et al. [34] designed a multi-modal, multi-robot interface and evaluated its performance based on operator workload. The study indicated that the majority of research currently done in human-robot interaction is on single-robot control interfaces for urban search and rescue environments. One difficulty of controlling a mobile robot is due to deficiencies in the interface design, which commonly employs multiple windows for displaying different types of information, such as video, map, sensor readings and control options. This leads to cognitive overload on the part of the user.

It was found that a single-window interface would allow for better robot control, and that the video display is primarily used in order to visualize the robot environment for

teleoperation. The video display, however, still causes control difficulties if used without other sensor data. In addition, although the user has good situational awareness (SA) for one robot, he or she has incomplete SA for other robots in a multi-robot system.

The proposed interface addresses the above problems by allowing the user to view the current robot in a large main video display, while keeping other robots visible in mini video displays. Also, the interface uses a mini-map in order to see the relative positions of all the robots. The interface provides the user with multiple input schemes, such as through the use of keyboard, mouse and voice. Experimental results demonstrate the viability of the interface.

Despite the relative increase in cognitive load due to managing multiple robots, the study found that the interface was able to allow the user to take advantage of multiple mental resources, as opposed to overloading one resource (i.e. such as vision). The user can give commands to one robot, while thinking about the location of another. The user can control the current robot using the main display, but at the same time have peripheral awareness of the other robots in the mini displays.

The experiments were conducted in simulation only. User workload was measured using NASA-TLX evaluation method. One limitation mentioned in the study was that the simulation environment was relatively simple. Additional research was needed to demonstrate the capability of the interface in more complex and real environments.

2.3.4 Scene Manipulation

This section discusses work done in relation to modifying the view or scene of the robot from a user interface, in order to command and control the robot. The scene can either be an augmented reality display or a map of the robot's environment.

Correa et al. [35] discusses the development of a framework for interaction with an autonomous robot forklift, where an operator can control the robot via a handheld touch interface. The interface allows the operator to communicate via speech or sketch. The user can draw on a canvas, which uses augmented reality to visualize the forklift with object and obstacle information from the environment. The robot uses audible and visual annunciations to convey its current state and intended actions. The system also provides a manual override option, where the user is able to enter the forklift and control it directly. Tablet interface currently designed to control a single robot, but can be extended to switch between multiple robots.

Kim et al. [36] developed a vision-based human augmented mapping system which uses a color vision camera for exploration. User input is incorporated with the mapping, telling the robot where to go or to create user nodes (U-nodes), which tell the robot to capture omnidirectional images and annotate the location with semantic information for users.

2.4 Summary

Much work has been done in developing highly autonomous mobile robots. However, more sophisticated algorithms that enabled better navigation in complex environments have higher computational complexity. Most systems make use of both a global and local planner to offset the computational load. Also, different sensing capabilities are combined to provide the robot with a richer representation of the environment.

Despite the advances in robot autonomy, a human operator is still needed for higher level tasks, which may not be performed efficiently by the robot on its own. Some tasks would require direct control by the user, although use of sensors and primitive robot navigation can aid in reducing user workload. In order to enhance user capability, he or she must

have good situational awareness (SA) of the environment surrounding the robot. Sensor fusion, 3D video and map views and augmented reality have been found to provide good SA, without increasing cognitive load to a very high scale. These schemes, although ideal for single-robot teleoperation, are insufficient for multi-robot control system.

For a multi-robot system, the user needs to have SA for one or more robots. Also, it would be extremely difficult for a single user to control multiple robots simultaneously. A semi-autonomous approach would reduce workload on the user in controlling multiple robots. Although much research has been devolted to single-robot control interfaces, some interface schemes have been proposed for multi-robot systems. One scheme is to have a main video display for the robot-in-focus, and smaller video displays for other robots. A map is also provided to visualize the relative positions of the robots with each other and with the environment. Another scheme is to use the map as the main display, while the smaller video displays are provided to visualize each robot's point-of-view.

Research has been done on overlaying virtual objects to either the map or the video display of the robot display. The path, goals and areas of interest displayed on the robot map, and the video display is augmented with sensor data. However, not much work has been made in developing interfaces that would enable the user to change details of the map in order to improve robot navigation, which is the goal of this work.

CHAPTER 3: PATH PLANNING ALGORITHMS

Since the mobile robot would have to move autonomously most of the time, it was necessary to find a good path planning algorithm that would be efficient and adaptable, particularly to dynamic changes to the robot's map. Path planning is a subset of the more general motion planning problem, whose fundamental task is converting a set of human-specified high-level tasks into a set of low-level movements (i.e. linear and angular motion) that a robotic system could execute [37]. Since motion planning can apply to various types of robotic motion (i.e. robotic hand grasping a cup), path planning is often used to refer to the task of creating a set of steps for a mobile robot to move from a starting configuration to a goal configuration.

A path planning strategy was needed that would enable a mobile robot to move in an indoor environment, such as hallways, without colliding into any obstacles, on its way to one or more waypoints, also known as goals. The map would define the movable space, as well as the location, size and orientation of obstacles. It is possible that some obstacles would be unmapped. The robot may or may not be able to avoid these obstacles, depending on whether or not its sensors are able to detect it. In the case where an obstacle is not detected, a human user should be able to mark a position on the map where the user thinks the obstacle is located. We would need a user-interface that would allow the user to modify the map, and send the updated map to the robot in real-time.

The path planning strategy would then consist of two activities: (1) autonomous navigation and (2) human-robot interaction.

For autonomous navigation, the following path planning algorithms were considered.

3.1 Potential Fields

In this method, the idea is to represent a point in the map as a point in a potential field, which represents both obstacles and a goal [38]. Potential fields are arranged on the map so that obstacles would tend to repel the robot, while the goal would tend to attract it. The combination of both repulsion and attraction in the potential field guides the robot towards generating a path that would reach the goal.

The obstacles were represented using an obstacle function, where each obstacle is represented as a Gaussian function having unity height. The center of the Gaussian function represents the center of the obstacle, and its variance, the width or diameter of the obstacle. The goal is represented using a goal function. The obstacle and goal functions take as input a coordinate (x,y) in the case of a 2D map, and output a resistance value (J).

For points that are nearer the obstacles, the obstacle function would output larger J values, while points nearer the goal, the goal function would output lesser J values. The sum of the outputs of the obstacle function and the goal function is what the robot uses to generate a path towards the goal. The robot can determine the next step to take by computing the resistance values around it and choosing the one that has the least resistance.

The advantage of this method is that the computation of the trajectory towards the goal is fairly simple. The path planning is done locally within a certain sensing radius. However, local minima may exist, where the robot would be unable to reach the goal area, which is considered as the global minima. If the robot is in a local minimum, and is unable

to find another configuration within its sensing radius that would lead it the goal, then it is trapped. The robot may move from one position to another nearby position, but never reach the goal.

A MATLAB simulation was set up in order to test the algorithm. The path planning algorithm using potential fields was adapted from [38]. An obstacle function was created that would model walls linear potential fields on a two-dimensional Cartesian map. A graphical user interface (GUI) in MATLAB was created in order to interactively add a start position, waypoints and obstacles to the map. The GUI enabled a user to change the map view, start or reset the simulation, and enable or disable a contour map view of the potential field.

Figure 3.1 shows the GUI after a path planning simulation has been run. Two waypoints had been added to the map, but no obstacles were added. The start location is at the bottom left (green start), while the goals are at the middle and top right of the map respectively. Note that the order of goal placement is important. Path planning to each goal is done in first-in, first-out (FIFO) order. The path to each goal is computed separately, which can be seen from the map. The path color from start position to goal 1 is blue, while from goal 1 to goal 2 is green. The total elapsed time to compute both paths is 10.51s.

Figure 3.2 shows a path planning simulation with two waypoints and one obstacle. Instead of making a path along the east hallway from the start position, the algorithm chose to make a path along the north hallway. The obstacle was able to influence the change in path.

Figure 3.3 shows how the algorithm can fail to reach the goal due to local minima. The robot enters a local minimum near the obstacle and becomes trapped. There are many ways to get escape, or even prevent, falling into a local minimum. One method is to change the obstacle and goal functions in order to minimize the occurrence of local minima.

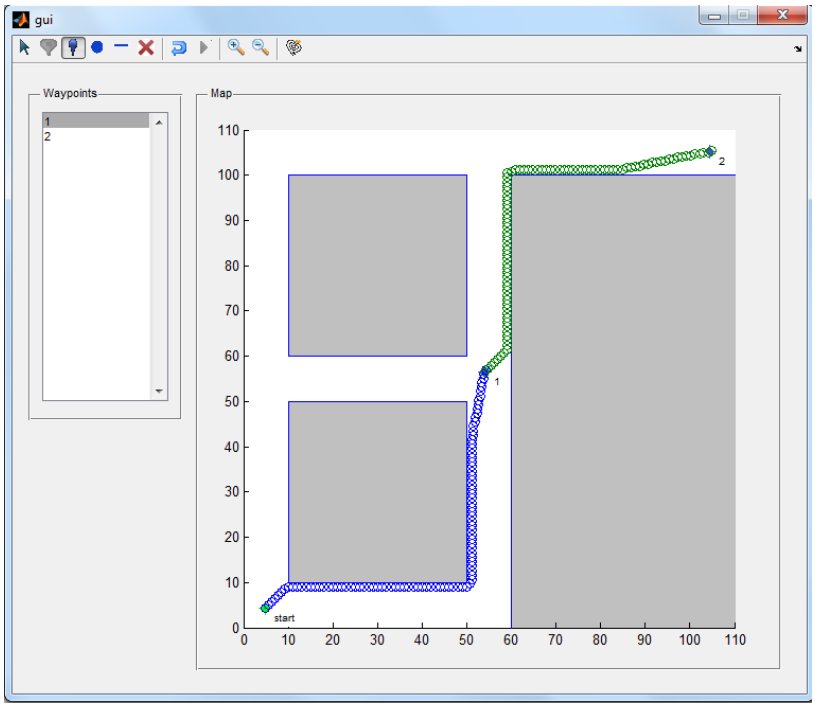


Figure 3.1: GUI depicting a path planning simulation using potential fields, where the map has two waypoints.

Another would be to create a look-ahead algorithm, that can plan several steps ahead in order get the best possible path outside of a local minimum. Another option would be to add mechanisms that would allow the robot to escape from local minima. The two latter methods would increase code complexity.

The computation time using the potential field implementation was found to be unacceptable. With local minima error correction, path planning may take even longer. Although the algorithm could be run much faster by pre-computing all the J values in the map before path planning begun, it was decided that another algorithm may be suited for the current problem, which would avoid local minima traps.

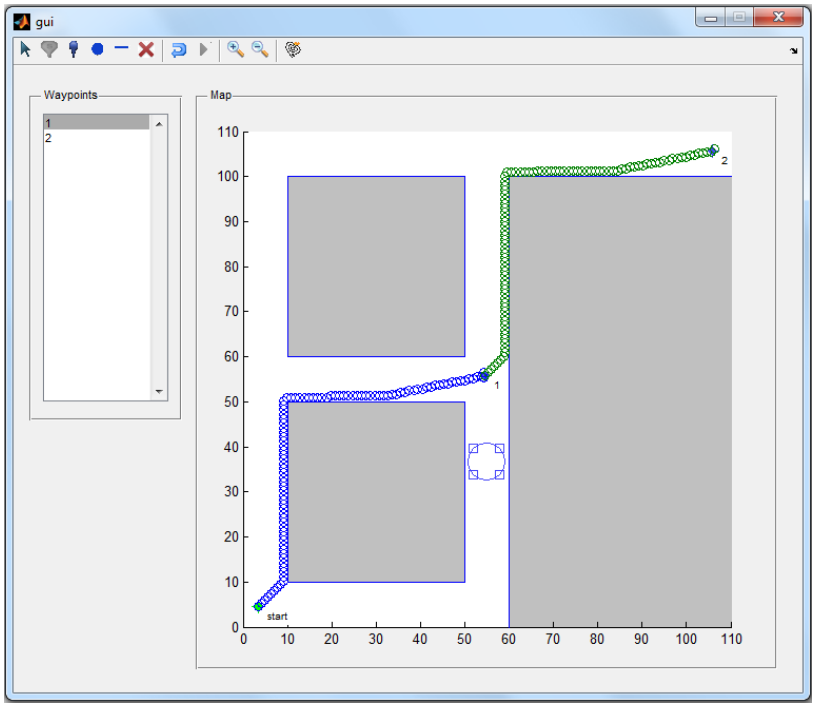


Figure 3.2: GUI depicting a path planning simulation using potential fields, where the map has two waypoints and one user-defined obstacle.

3.2 Probabilistic Road Map

As an alternative to the potential field method, the probabilistic road map (PRM) method was considered. PRM is a sampling-based algorithm that generates a set of free configurations, which could be represented as points on the map where the robot would be able to path. The algorithm can then create a path from start to goal configuration by connecting the free configurations. The algorithm starts by taking random samples from the configuration space of the map. [39]. Then, it checks whether each sample belongs to a free configuration space, which is the space where a robot can pass through, or not. Samples that are taken on or within obstacles do not belong to the free configuration space. Samples that are at a certain close distance from the obstacles may also be removed.

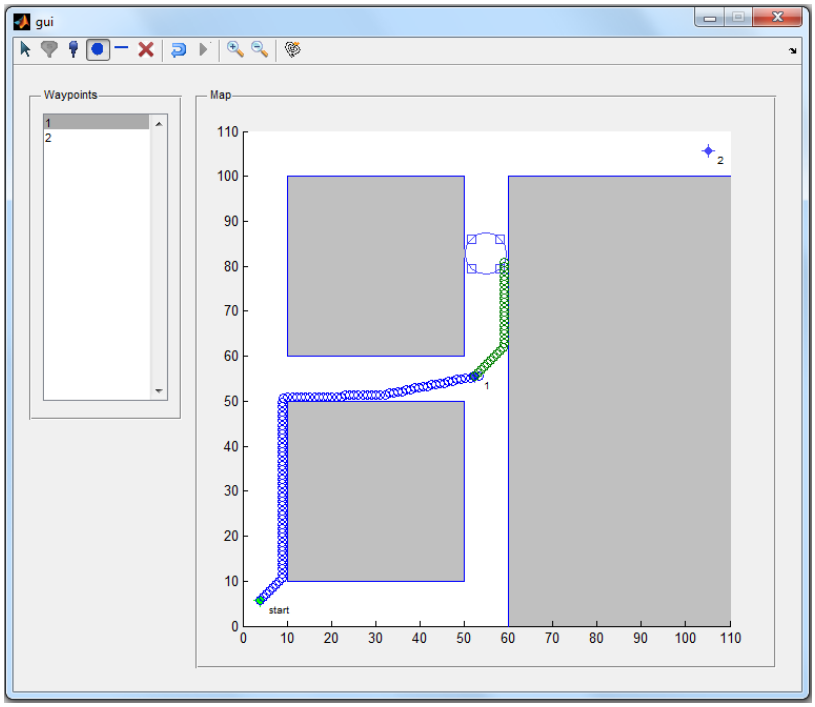


Figure 3.3: GUI simulation where planner was unable to reach the goal due to local minimum.

A local planner attempts to connect each configuration to its nearest neighbors. The start and goal configurations are then added to the existing connections, resulting in a graph. A path from start to goal can then be established through use of graph searching algorithms, such as Dijkstra's algorithm.

A MATLAB simulation was created in order to test the PRM algorithm. However, a GUI was not generated because testing could be done much faster using a script.

Figure 3.4 shows the path generated by the PRM algorithm. The start point is indicated by the circle in the bottom left part of the map, while the goal is the âĂŸX' at the top right. The total number of samples in the configuration space is 2500. The total number of free configurations generated for the above test is 2000, which are represented as blue dots on the map. The path is the blue line from start to goal.

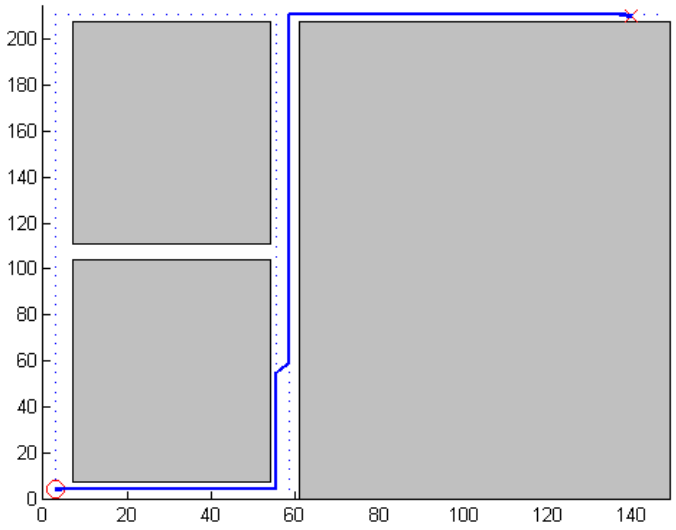


Figure 3.4: PRM simulation using random samples from free configuration space.

The graph was generated by computing the distance from each free configuration to every other free configuration, and storing the distances in a 2D matrix, such that the distance specified by a row and column of matrix represent the weighted edge between one free configuration node (row index) and another free configuration node (column index). The shortest path can then be computed using Dijkstra's algorithm.

Table 3.1 shows the performance results for the PRM algorithm. It took almost 6 seconds to generate the free configuration space. The reason is that each sample is tested to see if it falls into an obstacle or not. The sample is randomly chosen from the set of samples comprising the configuration space. The more obstacles there are in the map, the longer the check will take. The map used has three rectangular obstacles representing the walls and closed spaces of a building. Generating the distance matrix took approximately one-tenth of a second due to the use of a MATLAB function, *pdist*, that can perform vector computations. Dijkstra's algorithm took 1.22s to generate a path.

Table 3.1: Computation times of PRM using random samples in the free configuration space.

Task	Elapsed Time (s)
Generate collision-free samples	5.71
Node distances calculation	0.10
Dijkstra's algorithm	1.22

The algorithm still needed to be optimized, as the 5.71s computation time for generationg collision-free samples was unacceptable. Instead of generating the free configurations randomly, they could be taken from a grid of points that represent the configuration space. The configuration space is overlayed with each obstacle such that points that fall within the obstacle are "zeroed out." The free configuration space is then the matrix of values containing non-zero samples (x, y).

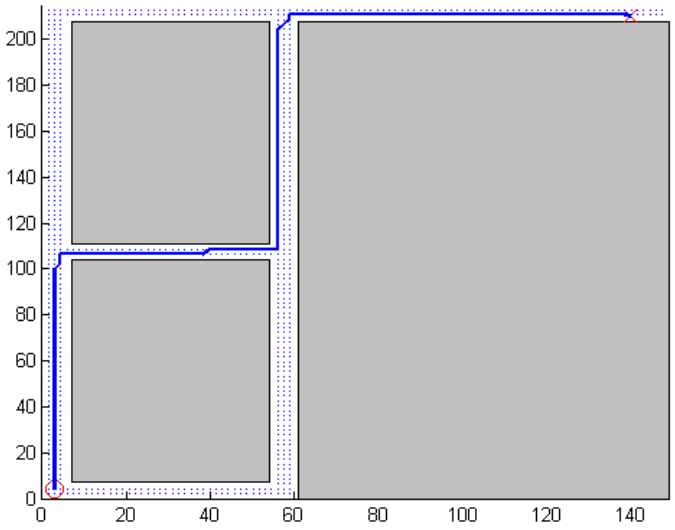


Figure 3.5: PRM simulation use grid of samples, where samples that fall within obstacles are excluded.

Figure 3.5 shows the same map using the grid-based PRM algorithm. The number of free configurations is 924, which are represented by the blue dots. The dots appear denser

than in the original PRM algorithm since the configuration space had 10000 samples. However, most of the samples were located within obstacles and were thus excluded from the set of free samples. Increasing the number of samples would make computation longer, but would lead to a smoother path.

Table 3.2: Computation times of PRM using fixed samples in the free configuration space.

Task	Elapsed Time (s)
Generate collision-free samples	7.75e-003
Node distances calculation	2.30e-002
Dijkstra's algorithm	2.54-001

Table 3.2 shows the computation times for the modified PRM algorithm. The time to generate the free configuration space is significantly lower than in the original algorithm. Since the number of free configurations is also lower, the times for both calculation of the distance matrix and Dijkstra's algorithm had improved.

Use of a grid-based structure for generating the free configuration space greatly improved performance compared to the potential-field based approach. Although the current algorithm was sufficient for planning a global path on the map, a much faster algorithm may be needed for more complex environments.

3.3 PRM-A*

The A* algorithm is a heuristic-based graph search algorithm that has properties similar to Dijkstra's, but uses a heuristic (normally represented as a function) in order to improve the speed at which a path from start position to goal position is found. The algorithm is designed to minimize the cost generated by traversing an edge and the cost generated by the heuristic function. For path planning, the Euclidean distance between one point (x,y) in the map to the goal was used as the heuristic.

The GUI shown in Figure 3.1 was redesigned in order to work with PRM. It was modified so that the user could change the ordering of the waypoints.

Figure 3.6 shows the path generated by the PRM-A* algorithm. The start position is the green point at the lower left corner of the map, while the goal is at the top right. The simulation generated 924 free configurations like the earlier PRM algorithm.

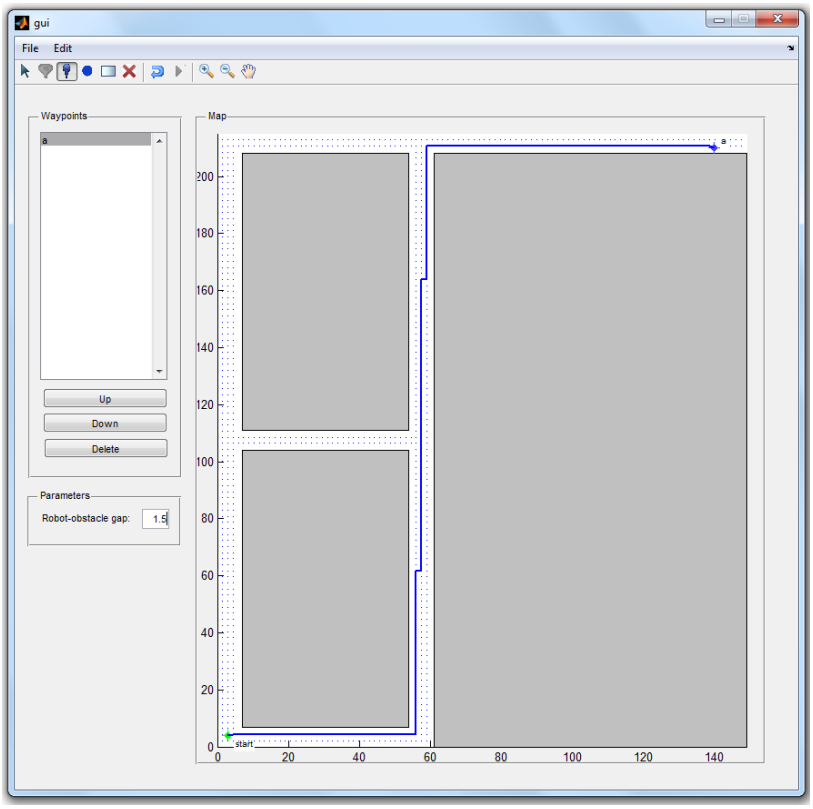


Figure 3.6: GUI-based path planning simulation using PRM and A* algorithms.

Table 3.3 shows the performance results of the PRM-A* algorithm. It did not perform any better than the standard Dijkstra's algorithm, given the same map, which was somewhat unexpected. The cause may have been that the A* algorithm also had to calculate a heuristic matrix of distances from each node to the goal, which contributed to the total computation time of the algorithm. There were other factors to consider, such as contention with resource on the computer, task scheduling, etc.

Table 3.3: Computation times of PRM-A*.

Task	Elapsed Time (s)
Generate collision-free samples	7.76e-003
Node distances calculation	3.13e-002
Dijkstra's algorithm	2.66-001

Figure 3.7 simulates a map with two goals and four user-defined obstacles. The algorithm 886 free configurations, a was able to produce a path around the obstacles, in order to reach the goal. The strength of the PRM algorithm, compared to the potential field-based path planning algorithm, is that it does not get stuck in a local minima. The PRM algorithm is a global planner, which takes plans a path by considering all of the possible configurations. The potential-field algorithm, however, is a local planner that can only plan a path within a given sensing radius.

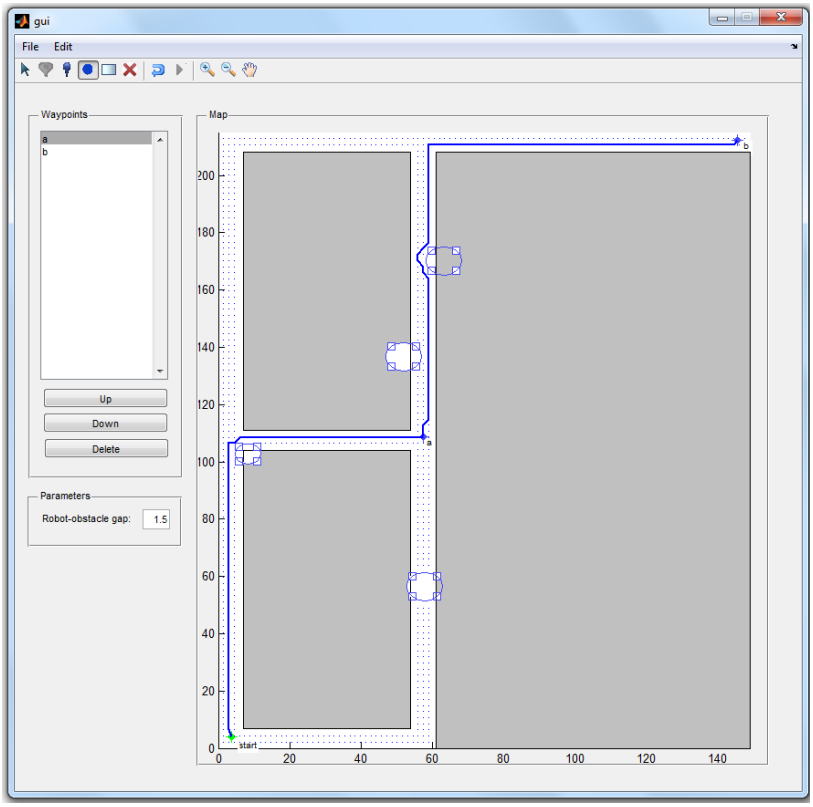


Figure 3.7: GUI-based path planning simulation having two waypoints and four obstacles.

In Table 3.4, adding more obstacles did increased the time for generating the free configuration space, however, it reduced the node distance calculation and path calculation due to having less free configuration samples.

Table 3.4: Computation times for PRM-A* path planning with two waypoints and four obstacles.

Task	Elapsed Time (s)
Generate collision-free samples	9.47e-003
Node distances calculation	2.88e-002
Dijkstra's algorithm	2.01e-001

The A* algorithm did not significantly improve the path planning computation over the Dijkstra's algorithm. However, it could still potentially become much faster with a proper specification of the heuristic function, that takes into account more information regarding the map and the path planning task.

Based from the tests, it was found that the PRM-A* algorithm would be sufficient as a global planner for the robotic system. The potential field-based implementation performed considerably slower than the PRM implementation, and suffered from local minima errors. However, having a globally planned path is not enough, as the robot has to take into consideration potential changes to its surroundings. A navigation algorithm must also have local planner that would detect changes to the environment and replan a local path accordingly. Also, the actual control inputs, such as linear and angular velocity, need to be specified programmatically.

CHAPTER 4: ROBOT SYSTEM

In deciding on a robotic system to use, one was needed that would be easily programmable, modular and reliable. The robot also had to allow for manual control (i.e. teleoperation). We chose the Pioneer 3-DX research robot from Adept MobileRobots.

4.1 Robot Hardware

The Pioneer 3-DX is a wheeled mobile robot that comes fully assembled, and can accommodate a number of accessories. It is intended to be used for research in an indoor environment. It has a high payload capacity (25kg), which allows it to accommodate various accessories. Accessories can be connected and powered through the robot's auxiliary serial ports. The robot platform is depicted in Figure 4.1. The robot specifications are summarized in Table 4.1.

Although the platform comes pre-built, it still needed another computer in order to operate high level functions. The computer can be integrated into the platform in one of the following ways: connect an on-board computer (provided by Adept as an accessory) directly into the platform, connect a serial-to-WIFI bridge, which allows the robot to communicate wirelessly to a remote computer, or connect a laptop computer, which sits on top of the robot platform ("piggyback").

Although a dedicated on-board computer would be convenient, it would be an added cost to the system. Use of a serial-to-WIFI bridge would also be a convenient option,

Figure 4.1: Adept MobileRobots Pioneer 3-DX Mobile Robot.

however, network reliability might be an issue, since the computer has to send "ping" messages at specific time intervals to the robot. It was decided that a laptop would be mounted on the robot platform and connected to the latter via a RS-232 serial cable. The on-board laptop specifications are summarized in Table 4.2.

The platform can also interface with a number of accessories, such as extra sonar sensors in the rear, laser range finder, color video camera, etc. At the minimum, the robot system needed a color video camera that would be able to send video to a remote computer. In order to reduce cost and development time, the laptop's built-in webcam was used for color video input.

The hardware setup is shown in Figure 4.2.

4.2 Hardware-Software Deployment

The robotic system has a three-tier control structure, consisting of the robot micro-controller, the onboard laptop and a remote computer, as seen in Figure 4.3.

Table 4.1: Pioneer 3-DX specifications.

Microprocessor	Embedded 44.2368 MHz Renesas SH2 32-bit RISC microprocessor with 32K RAM and 128K FLASH
Motors	2 with 500-tick encoders
Sensors	8 sonar (forward facing)
Batteries	3 hot-swappable (12VDC, 9Ah)
Ports	4 RS-232 serial ports
Body	Aluminum
Wheels	2 foam filled, knobby tread, 195mm diameter, 47.4 width
Software	MobileRobots' Advanced Robot Interface for Applications (ARIA), written in C++
Length	44.5cm
Width	39.3cm
Height	23.7cm
Clearance	6.0cm
Weight	9kg
Payload capacity	25kg

Table 4.2: Laptop specifications.

Name	Dell Latitude E4300
CPU	Intel(R) Core (TM)2 Duo CPU P9600 @ 2.53GHz
OS	Ubuntu Desktop Linux 12.04 LTS
RAM	4GB

The robot platform's microcontroller provides the basic functions that control the velocity of each wheel. It also receives commands from, and sends sonar and wheel-encoder data to, the onboard computer. Communication follows a client-server relationship, where the robot acts as a server that can support multiple clients. The client must send periodic "keep-alive" messages to the robot server. Otherwise, the server will issue an emergency stop for safety purposes. The onboard computer can communicate to the robot server via the Advanced Robot Control and Operations Software (ARCOS) interface. AR-

Figure 4.2: Pioneer 3-DX with laptop computer.

COS is a low-level, packet-based protocol, which is implemented in the robot's firmware. The packets are bits of data sent directly to the robot's serial port.

The onboard computer provides high-level functionality to the robot. The robot comes with the Advanced Robot Interface for Applications (ARIA), which is a C++ software development kit. It provides high-level functions for advanced control of the robot. Note that the onboard computer can also function as a remote computer if the robot is fitted with a serial-to-WIFI adapter.

The remote computer is used to control the robot. It communicates with the onboard computer via ArNetworking, which is the networking component of ARIA. ArNetworking provides functions for tele-operating the robot, and handles all packet-based network com-

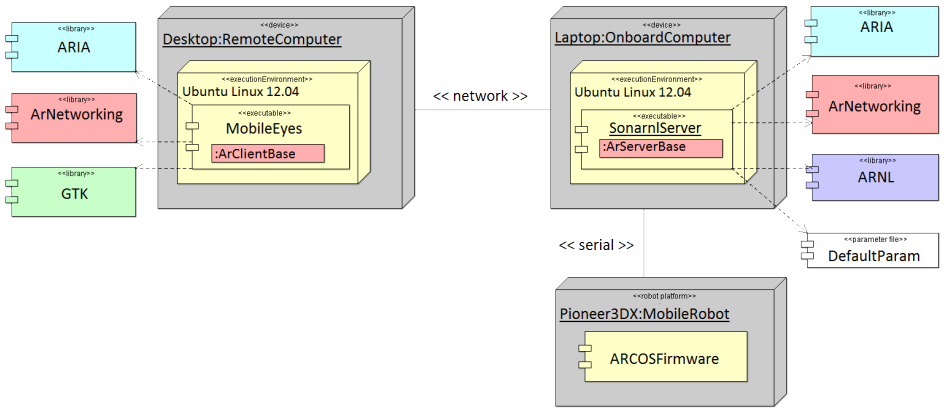


Figure 4.3: Robotic system hardware-software architecture.

munication through TCP/IP. More detailed information about the software stack can be found in Chapter 4.3.

4.3 Proprietary Software

The Pioneer 3-DX robot platform comes with its own software development kit (SDK). Additional software is also provided for interfacing with accessory components. The following are the software packages that were used.

4.3.1 ARIA

The Advanced Robot Interface for Applications (ARIA) is an open-source software library that provides high level functions for controlling the mobile robot. It handles all packet sending and receiving between the onboard computer, which it runs on, and the robot platform. Functions are provided to allow programmatic control of the robot (i.e. move at a certain speed, in a given direction), or manual teleoperation via keyboard or joystick. ARIA also handles interfacing with robot accessories, such as pan-tilt-zoom (PTZ) camera, laser

range finder (LRF) and robotic gripper. Adept provides such accessories, and the additional software needed to use them with the robot. ARIA is a native C++ library. However, interface wrappers are available for Java and Python.

4.3.2 ArNetworking

ArNetworking is the network communication component of ARIA. It uses standard TCP/IP socket communication to send control and information packets to and from the onboard computer. ArNetworking uses a client-server model, where the remote computer is the client and the robot platform's onboard computer acts as the server. Multiple remote clients can connect to the robot server. Also, a separate remote server can be run that would allow multiple robots to communicate with multiple clients. ArNetworking allows different level of control of the robot, ranging from manual teleoperation using keyboard or joystick, to advanced navigation using waypoints (which is performed using an additional software package).

4.3.3 ARNL

The Advanced Robot Navigation and Localization (ARNL) library is a set of software packages that provide the robot with intelligent navigation and localization. Navigation allows the robot to go from a start position to a goal position autonomously. Localization allows the robot to know where it is in the environment. Localization can be done using either sonar or laser sensors, if available. The navigation and localization algorithms of ARNL are closed source. However, APIs are provided to allow for a user to integrate his or her own navigation or localization algorithms.

4.3.4 SONARNL

SONARNL is the sonar localization package of ARNL. It provides a set of functions for localizing the robot using its sonar sensors.

4.3.5 MobileEyes

MobileEyes is the remote teleoperation and navigation client software for Adept robots (Figure 4.4). It has a graphical user interface (GUI) for viewing the robot remotely through a two-dimensional map, which is loaded from a file. It provides functions for controlling the robot in different ways: (1) teleoperating the robot using either a keyboard or joystick, (2) navigating the robot to a certain position in the map using a mouse and (3) commanding the robot to go to a goal position in the map (requires ARNL). It also provides a way to modify the robot's behavioral configuration (i.e. maximum velocity, obstacle-avoidance, etc.) in real-time. If the robot is interfaced with a video camera, MobileEyes can display live video streams. This allows for non-line-of-sight teleoperation.

MobileEyes uses the ArNetworking library to communicate remotely with the robot server. However, it is closed source and only the executable is provided.

4.3.6 MobileSim

MobileSim is a robot simulation software for ARIA-based robots (Figure 4.5). It allows a user to test and debug software used to control the robot. It has GUI for visualizing robot pose and its environment. It can display line data from a map file, which represents the map of the robot's environment. MobileSim is closed source. However, it is based on

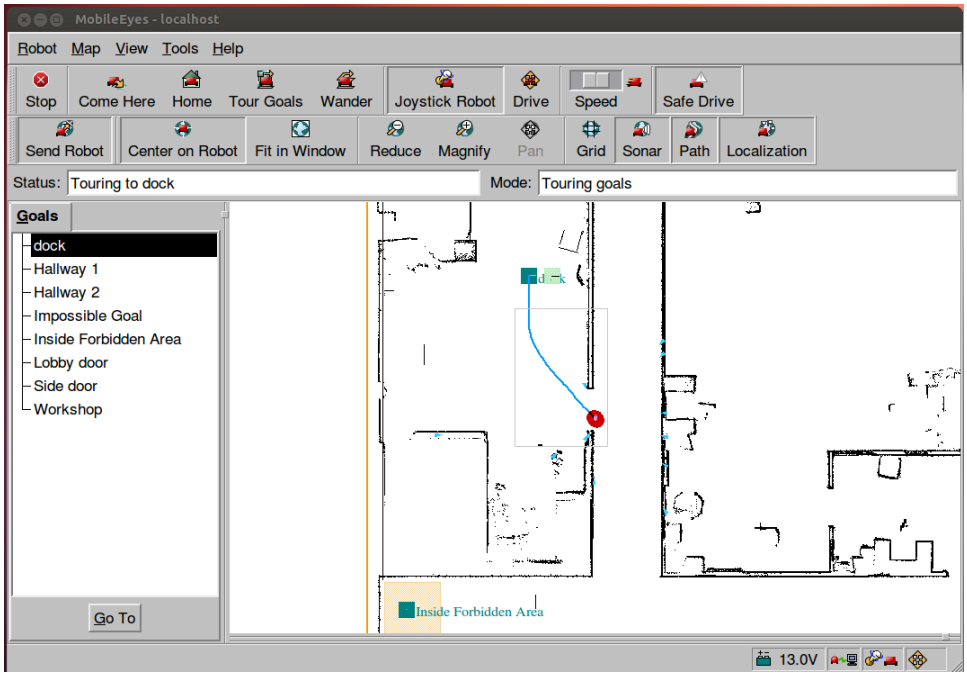


Figure 4.4: MobileEyes GUI.

the Stage simulator, which is part of the Player/Stage project, a free software tool for robot and sensor applications [40].

4.3.7 Mapper3Basic

Mapper3Basic is a software tool for creating maps, which are used by ARNL, MobileEyes and MobileSim for visualization and navigation (Figure 4.6). Maps consists of lines, points and rectangular regions. Lines represent obstacles (such as walls) in the map. Points can either be obstacles, or home and goal positions. Regions represent forbidden areas of the map. Maps are loaded from and saved to a map file (filename extension .map), which is a human-readable file format. Mapper3Basic is closed source.

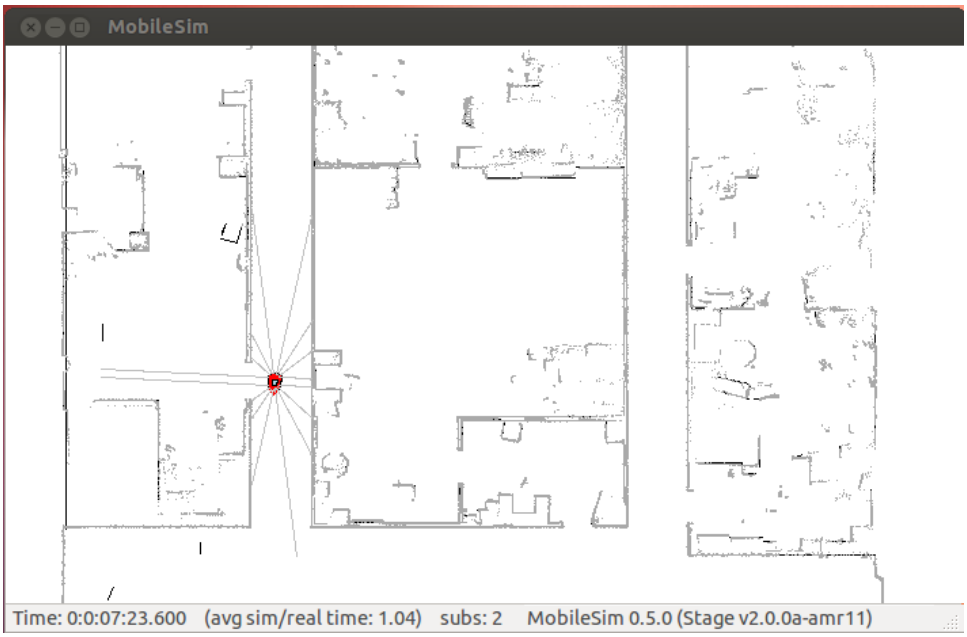


Figure 4.5: MobileSim GUI.

The more advanced map editor software for creating maps is Mapper3, which allows for point cloud-based mapping using a laser-range finder. This software is only available with the corresponding LRF accessory provided by Adept.

4.4 System Modifications

The proprietary software package had several features that performed most of the high level control operations needed in the study. However, it still lacked some features that were critical components of the study. The software needed to be modified in order to support dynamic map manipulation. Also, the software package had its own navigation algorithm, which provided both global and local planning. Although a path planning algorithm was already designed, as explained in Chapter 3, it could only determine a path to reach the goal, and not the specific inputs that the robot needed to make in order to reach the goal. In order to reduce development time, ARNL was used as the navigation

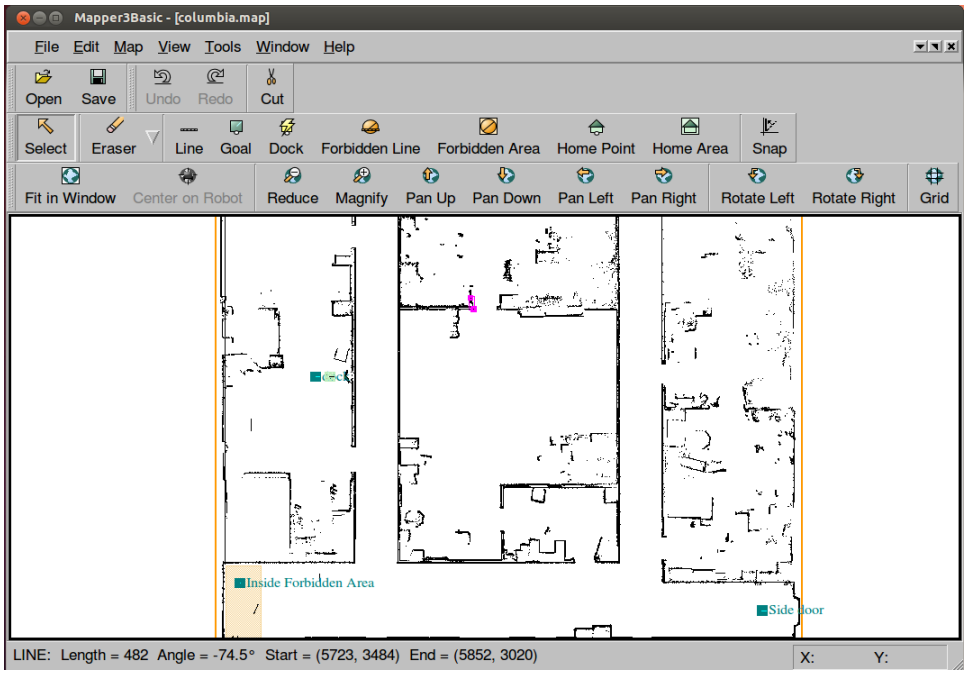


Figure 4.6: Mapper3Basic GUI.

algorithm. The algorithm's parameters were modified in order to improve performance in the test environment.

Figure 4.7 above shows the changes made to the original hardware-software architecture. The following subsections describe the changes in more detail.

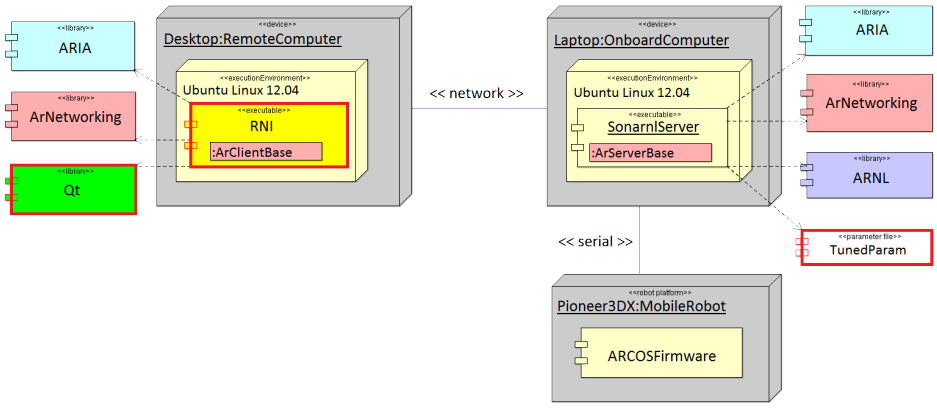


Figure 4.7: Modified hardware-software architecture.

4.4.1 Navigation Algorithm

After several path-planning tests conducted with the robot using ARNL for navigation, it was found that the algorithm was sufficient for the study. The disadvantage is that the algorithm is closed source, and insufficient information is given regarding the design of the algorithm. However, ARNL provides an API that allowed us to modify certain parameters to change the behavior of the robot. The navigation algorithm was optimized so that the robot would perform well in a narrow hallway.

Table 4.3 describes the ARNL path planning parameters that were modified, and the reasons for their modification. Table 4.4 shows the parameters that were considered but left to their default settings.

The default localization and path planning parameters used by ARNL did not allow the robot to move toward the goal with an acceptable speed. Preliminary testing showed that the robot would not reach the goal in certain cases, such as when the goal is placed near a wall. The path planning parameters were optimized so that the robot could navigate through a narrow hallway while avoiding box-type obstacles.

4.4.2 Robot Navigation Interface

Although MobileEyes could perform most of the required navigation tasks, it was missing one critical feature that was needed in the study - allowing a user to update the robot map in real-time. Since the software was closed source, it would be very difficult to modify it to support the missing feature. Instead, a custom navigation software, called Robot Navigation Interface (RNI), was created. RNI used both the ArNetworking and Qt GUI framework libraries.

The interface as shown in Figure 4.8.

Table 4.3: Modified path planning parameters.

Parameter	Default	Current	Description	Reason for change
PlanRes (mm)	106.25	49.125	"Resolution of the grid used for path planning"	Lower value means higher resolution, which would allow the robot to move in tighter spaces.
PlanFreeSpace (mm)	637.5	98.25	"Preferred distance from side of the robot to the obstacles"	A lower value is needed so that the robot can traverse through tight spaces surrounded by obstacles.
UseCollision RangeForPlanning (bool)	False	True	Enabling this flag forces the robot to sense as far as the max distance of the sensor in order to plan paths around the obstacle	Enabled so that the robot can replan a path as soon as it encounters a possible obstacle
GoalDistanceTol (mm)	200	10	Distance to the goal (in mm) that would be considered as "reached"	Lower value is preferred so that the robot will be completely within the goal marker
GoalSpeed (mm/s)	250	350	"Maximum speed at which end move to goal is executed"	Increased the value so as to increase the speed at which robot moves to the goal
GoalRotSpeed (mm/s)	33	50	"Maximum rotational velocity at which end move to goal is executed"	Same reason as for GoalSpeed
NumSplinePoints	5	3	"Number of points that will be used to subdivide the look ahead in the local path which will then serve as the knots for the spline to form over."	Reduce computation time and reduce the amount of turns that the robot needs to make by making the path more linear.

Table 4.4: Path planning parameters that were not modified.

Parameter	Default	Description	Reason for no change
MaxSpeed (mm/s)	750	Maximum speed during path following	From various tests, the robot does not reach 750mm/sec while traversing through obstacles in the map, so changing the maximum speed limit was not necessary.
MaxRotSpeed (deg/s)	100	Maximum rotational speed	Testing is done mostly in maps that require the robot to move linearly, so increasing this speed was not necessary.
CollisionRange (mm)	2000	"The distance from the robot within which the obstacle seen by the sensor and those on the map are used to compute the local path"	Since sonar sensing becomes more inaccurate as the distance from the robot increases, increasing this value might make lead larger errors in obstacle detection and path planning.
UseSonar (bool)	True	If true, use sonar for collision avoidance	This is necessary in order to avoid colliding with unmapped obstacles.

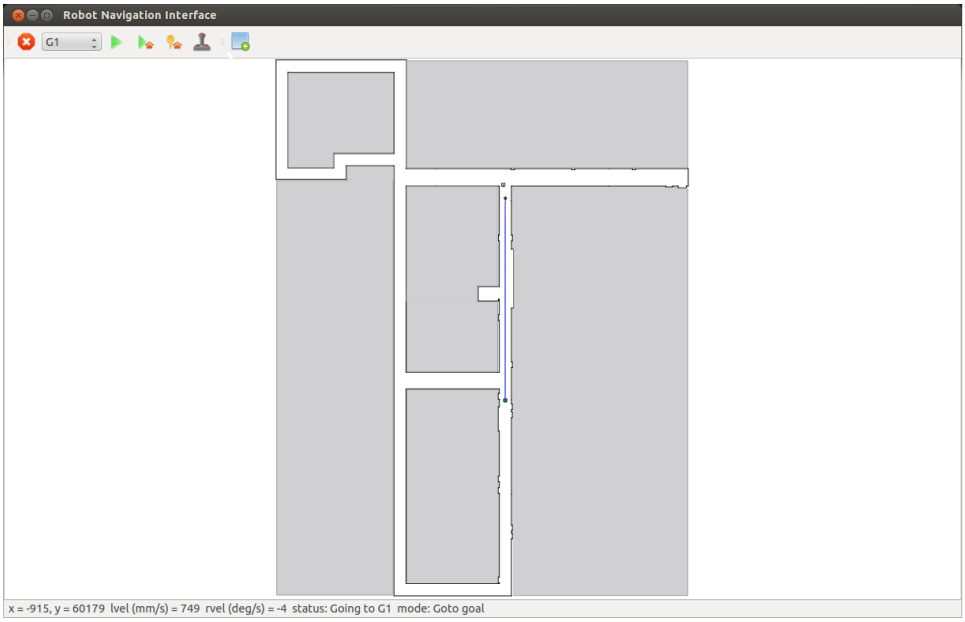


Figure 4.8: Robot Navigation Interface (RNI).

The new interface allowed the user to add obstacles to the map and issue map changes to the robot in real-time. Also, it had similar features found in MobileEyes, such as sending the robot to a goal, stopping the robot, resetting the robot to its starting position, and teleoperating the robot using the keyboard. It also shows the robot's path and obstacles on the map.

In order to add obstacles, the user must toggle a button on the toolbar and left click on the map with a mouse. The user can add as many obstacles on the map as needed. However, the new data is not sent to the robot until a right click on the map is performed. This allows the user to add multiple obstacles, and send the new information to the robot once, which results in better performance than sending each obstacle to the robot one at a time.

CHAPTER 5: TEST METHODOLOGY

The maps used in the tests described in the following sections were generated using Mapper3Basic. The base map in which all other test maps are derived was modelled after the second floor of Engineering Building II (ENB) of the University of South Florida. Since a laser range finder was unavailable, the map was created by hand and manually encoded using Mapper3Basic. The floor of the hallway was made up of square tiles, which helped in computing the lengths of walls and relative positions of the obstacles in the map. Landmarks, such as doors and windows, were also used to aid in accurate map creation. It was necessary to have a map with known landmarks, even if these landmarks would be unmapped, as these would be used to guide placement of robot and obstacles.

5.1 Test Setup

Two types of tests were set up: a simulation test and a real-world test. The simulation test was used to calibrate the robot's navigation parameters and test the Robot Navigation Interface (RNI) with map manipulation. The real-world test was done to verify the proposed map manipulation method. There are a number of conditions that cannot be simulated with the current system set up, such as live-video streaming, encoder errors when the robot is moving, and sensing of unmapped obstacles.

5.1.1 Simulation Test

The simulation test were performed using MobileSim. The entire setup could be performed on just one computer. However, the three-tiered control architecture was still needed. First, MobileSim was executed, which is similar to turning on the robot platform. Next, the robot server application, which uses ARIA, ArNetworking and ARNL and SONARNL libraries, was ran. The robot server application was based from an example code included in the ARNL software package. The application connects either through serial port (default), or through network connection by giving it a "host" parameter. The application automatically detects if a MobileSim simulation (which is usually identified as "localhost"), and connects to it if there are no other specified connection methods. The map is loaded when the robot server application is invoked.

Lastly, the RNI software was executed. It was designed to connect to the robot server, which in simulation, is also identified as "localhost" but uses a different port. When it connects, it automatically requests the map that was loaded in the robot server. Before sending the robot to the goal, the robot needed to be localized to its home position first. This could be done using the "Reset to Home" feature of the software. In simulation, the robot can be reset to its home position very easily, since no physical counterpart need to be moved.

Figure 5.1 shows the interface setup for simulation tests.

When the "go to goal" command is sent to the server, the robot enables path planning and moves towards the designated goal. While the robot is moving, the user can modify the map by placing obstacles along the path, and the map updates are sent to the robot with a click of the mouse. The robot, after receiving the new map configuration, would then recalculate a path around the obstacle.

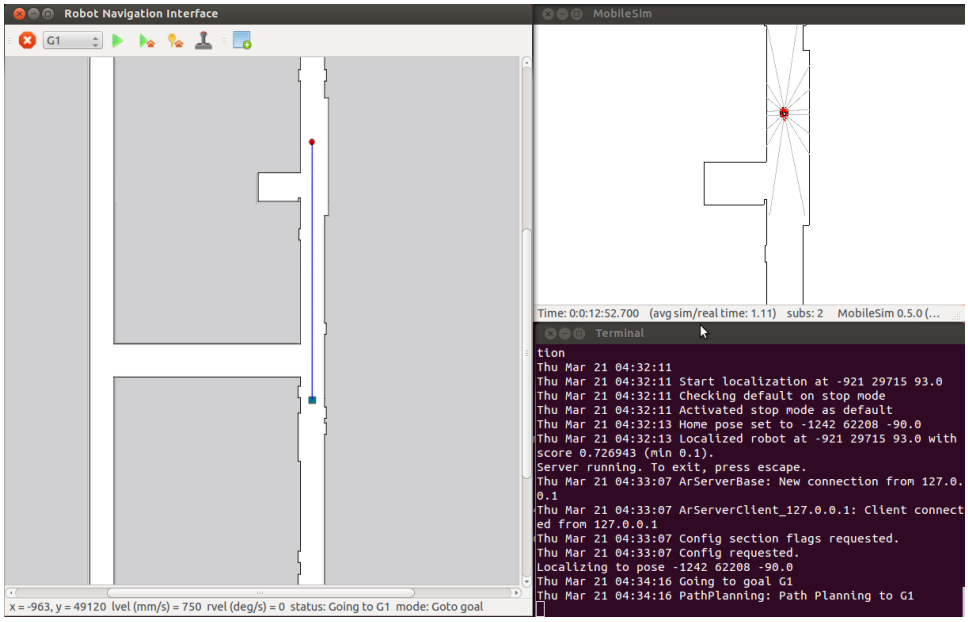


Figure 5.1: Remote navigation setup for simulation tests.

5.1.2 Real-World Test

In the real-world test, the laptop was placed on top of the robot platform and connected it to the robot via the RS-232 serial port. Unlike the simulation test, the real-world test will have a video of what the robot is "seeing" straight ahead. The laptop display was set at an angle that would allow the built-in webcam to get a forward view of the robot's environment. A third-party video chat application, was used to stream live video from the onboard laptop to the remote desktop computer.

Initially, the robot would be placed at the home position in real space corresponding to the position and of the home marker in the map. The heading was also matched with that shown on the map. The robot server application can be executed from the remote computer by using Secure Shell (SSH) [41] to connect to the onboard laptop.

RNI and the video streaming application are placed side-by-side so that the user can see both in one screen on the remote computer, as shown in Figure 5.2. In the future, these modules can be combined into a single application.

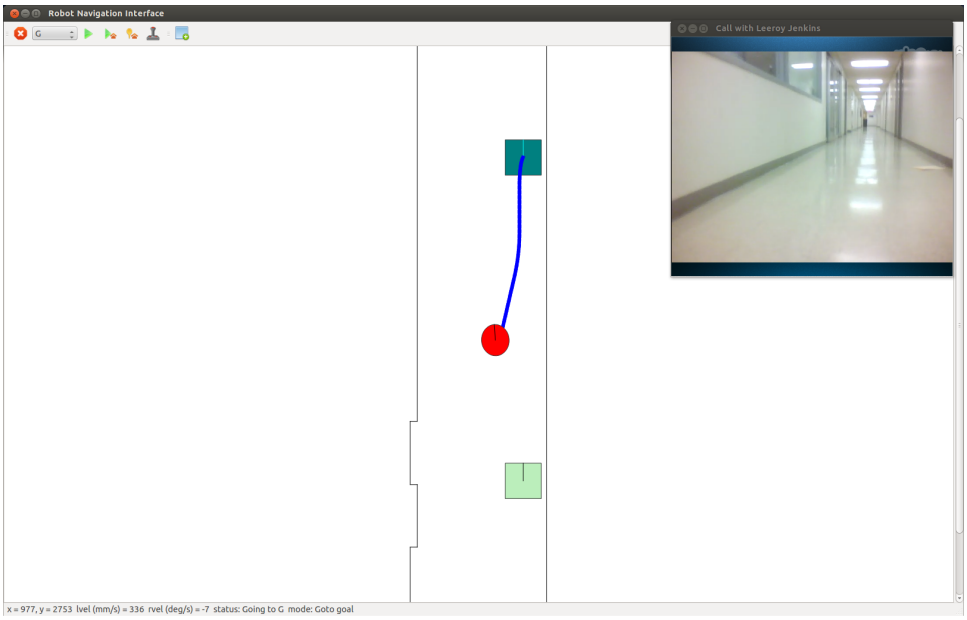


Figure 5.2: Remote navigation setup for real-world tests.

The same operations are performed as in the simulation. However, the difference is that there is a video display, which the user can use to detect any unmapped obstacles in the map.

5.2 Test Description

The following subsections describe the tests that were performed.

5.2.1 Navigation Test

The purpose of this test is to determine if the robot can navigate through an area with an unmapped obstacle much faster if the user marks the obstacle on the map in real-time. The virtual obstacle may not be placed in the exact location of the real-world obstacle, but the robot should still be expected to avoid it and reach the goal. Also, it is expected that the robot will plan a better path if the virtual obstacle is placed on the map when the robot is still far away from the target obstacle than when it is very close. If the virtual obstacle is placed as soon as the robot is started, then the performance should be comparable to the case where the real obstacle is mapped. Table 5.1 describes each test to be performed.

5.2.2 User Perception Test

This test seeks to determine how well the operator can map the location of an unmapped obstacle on the map by viewing the obstacle through either still images or a video stream. The user will perform both still-image and video-based tasks.

In the image-based mapping task, the user will go through a series of images, where the sequence emulates the motion of a robot towards a certain goal and an obstacle blocks its path. Each image shows the point-of-view of the robot at pre-defined and evenly distributed distances from the obstacle. The user must guess the location of the obstacle on the robot's map based upon the image. The RNI application is used for map manipulation. It stores the location of obstacles that the user places on the map. The user must place only one obstacle per image. After placing the obstacle, the user can move to the next image, until all the images have been viewed. The task can be done without the need of the actual robot. The user interface setup for the image-based mapping task is shown in Figures 5.6

Table 5.1: Navigation test description.

Test Name	Test Description
Simulation test with mapped obstacle	This simulation test was used as a baseline for robot performance. The obstacle is defined on the map, and the task of the robot is to go from start position to goal without hitting the obstacle. The user does not update the map.
Real-world test using teleoperation with one mapped obstacle	This test is used to see how an expert user would navigate the robot around the obstacle to the goal.
Real-world test with mapped obstacle	Similar to the simulation test, but with one obstacle defined on the map sent to the robot. This is used to compare simulation and real-world performance.
Real-world test with unmapped obstacle	This test is to check how the robot performs on its own when faced with an unmapped obstacle.
Real-world test with unmapped obstacle, single placement	In this test, the obstacle is removed from the map. Instead, the user has to place the obstacle in the position where the original obstacle was on the map. The user will do this by checking the video feed from the robot and making an estimate of where the real-world obstacle is located in relation to the map.
Real-world test with unmapped obstacle, multiple placement	This is similar to the previous test, however, the user is able to place multiple obstacles on the map to prevent the robot from hitting the real-world obstacle.

and 5.7. The images were taken at intervals of 3.05m. No time constraint was given to the user on placing the obstacle for each image.

In the video-based mapping task, the user must identify the location of an obstacle shown on a video stream, which depicts the forward point-of-view of the robot. This task is conducted with the real robot moving along a mapped area. Unlike the image-based task, the user can place as many obstacles on the map while the robot is heading towards the goal.

However, since the robot is moving constantly, the user has an effective time constraint on placement of obstacles on the map. The RNI application is used for data collection.

Two different mapped hallways are used for the test. The first hallway has multiple landmarks which allow the user to match what he or she is seeing on the image or video with the map. The second hallway also has landmarks visible on the video, but not on the map. The expectation is that the user should have an easier time identifying locations on the first hallway set up than the second because of the mapped landmarks.

The first hallway setup (Test 1) has two arrangements. One is where obstacle is placed in front of a corner (Test 1A), and the other is where the obstacle is placed behind the intersection (Test 1B). From the start position of the robot at one end of the hallway, the intersection is not very visible on the video stream or still image. However, as the robot moves down the hallway (either the physical robot is moving or the user is scrolling through the images), the intersection becomes more visible. The intention is to know how well the user would be able judge whether the obstacle is in front of the intersection or behind it.

The hallway for Test 1 is shown in Figure 5.3. Figures 5.4 and 5.5 show the obstacle placement in front of and before the intersection respectively. The interface setup showing the maps used in Test 1A and Test 1B and the starting images of the areas are shown in Figures 5.6 and 5.7 respectively.

The second hallway setup (Test 2) has one arrangement where the obstacle is between the robots start position and the goal. Here, the objective is simply to know how well the user can determine the obstacle's position with a few landmarks on the map. The second hallway is shown in Figure 5.8. Figure 5.9 shows the map and video interface setup for Test 2.

Figure 5.3: Hallway used in Test1A and 1B.

Figure 5.4: Obstacle placement for Test 1A.

Figure 5.5: Obstacle placement for Test 1B.

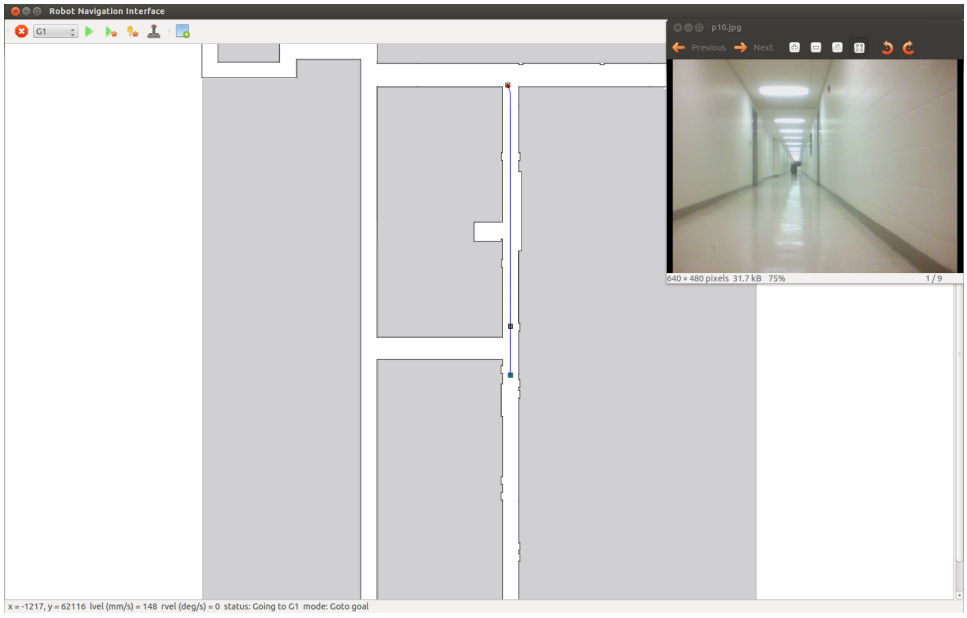


Figure 5.6: Image-based interface setup for Test 1A.

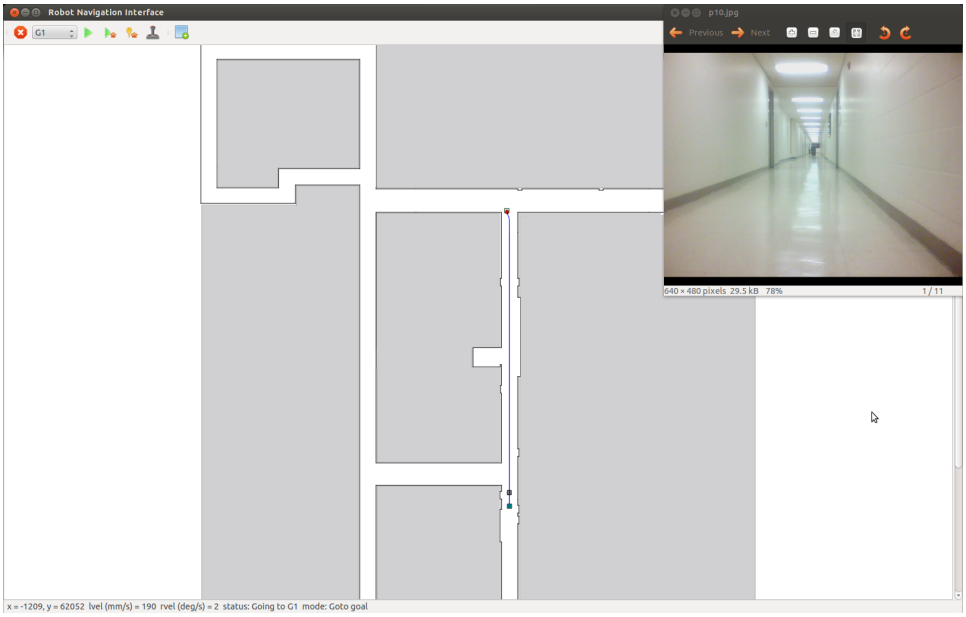


Figure 5.7: Image-based interface setup for Test 1B.

Figure 5.8: Hallway used in Test 2.

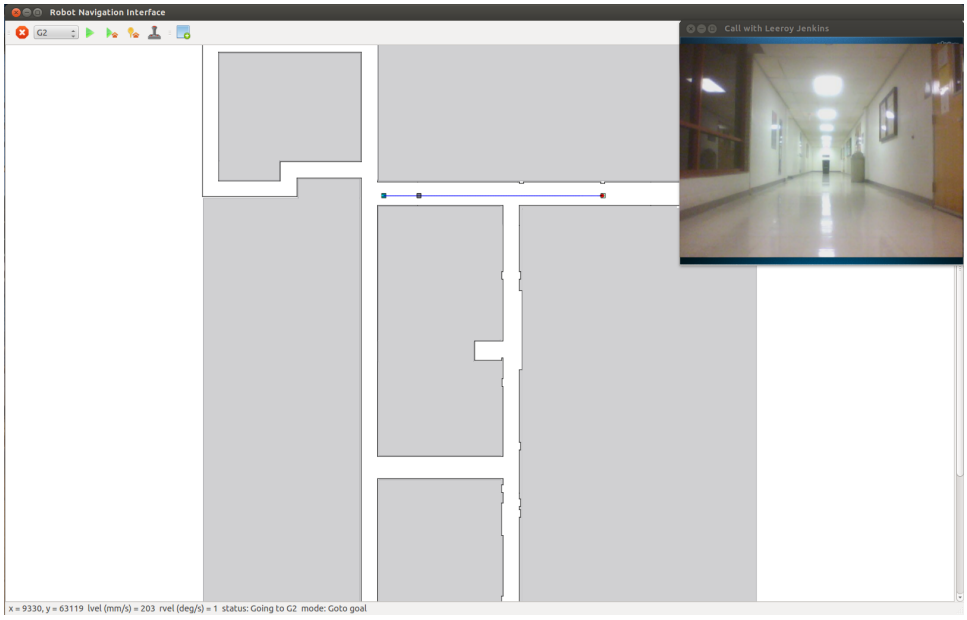


Figure 5.9: Video-based interface setup for Test 2.

The different tasks were performed by multiple users from the Department of Computer Science and Engineering in USF, where each user performed either Test 1A or Test 1B and Test 2 for the image and video-based setups.

The locations of obstacles that the users placed on the map was compared to the position of the real obstacle. The error e is obtained by getting the Euclidean distance between the real obstacle and the placed obstacles. The distance d of the robot from the real obstacle is also computed.

The expectation is that as the robot moves closer to the obstacle (lower d), the user should have a better grasp of the obstacles position relative to its surroundings based on the video, and can more accurately mark the obstacle on the map (lower e). Also, the user should be able to more accurately identify the position of the obstacle when there are multiple landmarks in map.

The participants were asked to complete a survey, which asked them for information that could affect their performance in the test. Specifically, they were asked their age,

college major or occupation, familiarity with the second floor of ENB, when they last had an eye exam, and whether they play video games or not.

CHAPTER 6: TEST RESULTS

6.1 Navigation Test Results

Table 6.1 summarizes the results of the navigation test. The elapsed times are computed from the moment the robot moves away from the home position (start time) to the moment that it reaches the goal with the correct final orientation (end time).

Table 6.1: Navigation test results.

Test name	Elapsed time (s)
Simulation test with a mapped obstacle	25
Real-world test using teleoperation with a mapped obstacle	17
Real-world test with a mapped obstacle	22
Real-world test with an unmapped obstacle	26
Real-world test with an unmapped obstacle, single placement	24
Real-world test with an unmapped obstacle, 3 placements	24
Real-world test with an unmapped obstacle, 5 placements	36

In the simulation test, the robot took a relatively longer time to reach the goal. The robot was able to reach the goal without hitting any obstacles, as expected. The real-world test with the robot being teleoperated by an expert user had the fastest time in the set at 17s. The user is familiar with the area, and both the map and video show the obstacle. With a mapped obstacle, the robot moved slower at 22s, which is 5s compared to the teleoperated case. However, the performance is 3s faster than the simulation equivalent.

The robot moved the slower at 26s with an unmapped obstacle and no user support. The robot was able to use its sonar sensors for obstacle avoidance. The robot performed 2s

faster in the unmapped case when the user was allowed to place an obstacle on the map. With multiple placement, the robot also performed the same as the single placement case.

The real-world test with an unmapped obstacle (no placements) took longer than that of the mapped obstacle case, which was expected since the robot would need to wait until its sonar hits the obstacle before it plans a new path. In the single placement case, the obstacle was placed immediately after the "go to goal" command was invoked, which should result in a scenario similar to the mapped case. However, compared to the mapped case, the single placement case was 2s slower. This could be due to the processing time required to update the map. Also, for safety reasons, the robot is stopped momentarily while its updating its map, which could be another cause for the delay.

The performance of the single and 3 placement cases are the same. This could be due to the fact that in the 3 placement case, the map update command is only sent to the robot once. Also, the obstacles were placed on the map in a way that does not hinder the robot's movement significantly (see Figure 6.1).

In contrast, the 5 placement case took the longest time at 36s. The obstacles were placed adjacent to each other and covered more than half of the hallway's width (see Figure 6.2). This made the robot move slowly in the area where the obstacles were placed, due to the smaller gap. The robot's navigation algorithm slows down the velocity in tight spaces.

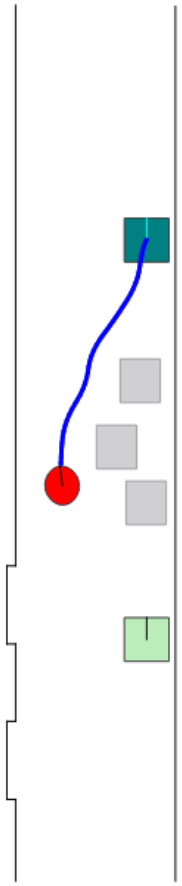


Figure 6.1: Real-world test with 3 obstacles placed separately.

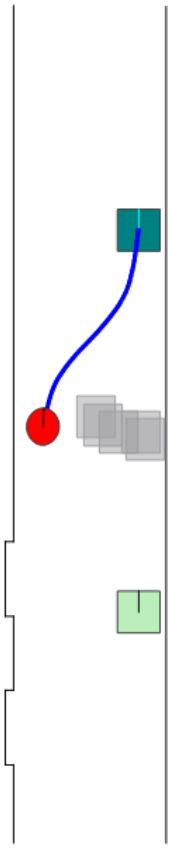


Figure 6.2: Real-world test with 5 obstacles placed overlapping with each other.

6.2 User Perception Test Results

The RNI application logs information regarding the robot's position when the user places an obstacle on the map and stores it in a file. It also logs time that the obstacle was placed, as well as the robot's status, linear and rotational velocity as well as the type of map that was used. In the still-image test, since no connection to the robot was made, the robot's position stored in the log file was initially set to (0,0). However, post-processing was done in order to register the proper robot position to corresponding obstacle placed on the map for each image.

The data was merged together with the survey information taken from each user into one CSV file. the CSV file was then imported to MATLAB for additional processing. The data was divided according to the type of test (image or video) and type of map (Test 1A, Test 1B, Test2). A plot of error e over distance d was made for each category. In every plot, e and d are both in units of meters (m). For the image tests, the obstacle placement errors at each position were averaged, and are shown on the plots.

A total of 21 people participated in testing. However, some data was discarded due to problems with test setup.

6.2.1 Image Test Results

Figure 6.3 shows a plot of the error in obstacle placement over the distance of the robot to the reference obstacle Test 1A. Most of the points appear to have an error within 8m. Also, it can be observed that there is a general increase in the distribution of the points as d increases. This was an expected result, since at relatively far distances, the user does not have a clear idea of depth in the image, and thus will be more prone to committing larger errors in obstacle placement. However, even at a distance greater than 25m, some

users were able to place an obstacle on the map very close to the reference obstacle ($e <$ 2m).

At $d > 20m$, the mean error follows a downward slope. At d = 27.15m, the mean error (2.74m) is lower by 0.71m compared to mean error (3.44m) at d = 8.84m. Apparently, most users were able to establish a relatively good guess on the obstacle's position even at far distances. However, on the next distance interval, most users changed their guess, which resulted in an error increase. The mean error peaks at $d = 21.05m$, where the highest error obtained is at 19.99m. However, as the distance becomes lower after $d = 21.05$m, the error progressively becomes smaller.

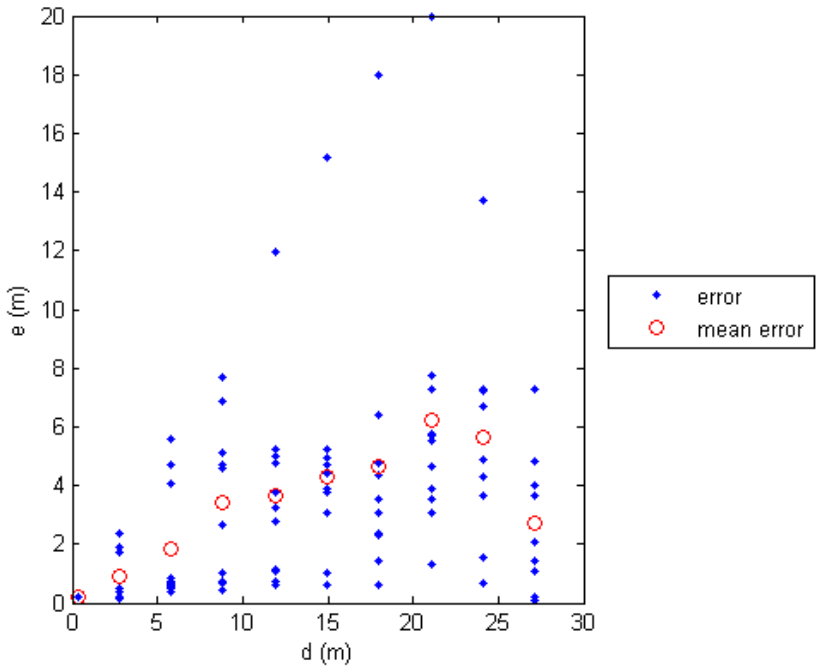


Figure 6.3: Error over distance for image-based Test 1A.

Figure 6.4 shows the data for the image-based Test 1B. The plot appears similar to the one in Figure 6.3. The same increase in the distribution of e values as d increases can be seen. A trend similar to Test 1A occurs for this case, except that there is now both a

downward and upward slope at the farthest distances ($d > 19.52m$). At $d = 25.62m$ the users were able to score better than at the next higher and lower distances ($d = 28.67m$ and $d = 22.57mm$ respectively). Getting the mean squared error (MSE) of the mean error with respect to 0 for both Test 1A and Test 1B, it appears that participants performed better in the Test 1B (MSE = 9.14 vs. 14.81). Note that the participants could perform either Test 1A or Test 1B, but not both. It's possible that the extra landmark (i.e. the intersection) provided a guide for users in Test 1B to better judge the position of the obstacle, as opposed to Test 1A where the obstacle was in front of the intersection.

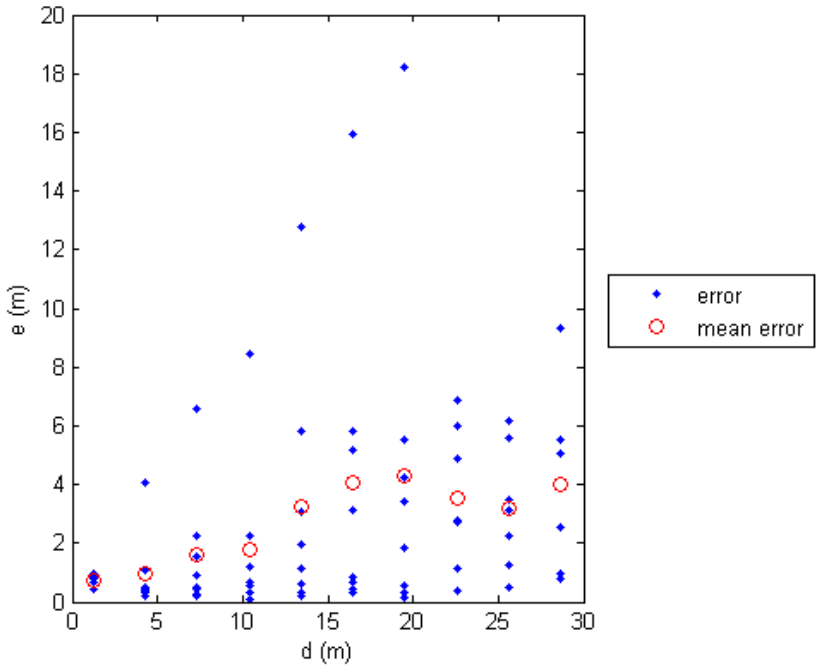


Figure 6.4: Error over distance for image-based Test 1B.

Figure 6.5 shows the image-based case for Test 2. Here, there are fewer intervals because the distance between robot's start position and goal was shorter. It can be seen that the mean error of the different distance intervals is much closer to each other than the previous image tests, which could indicate that most users were able to generate similar guesses

as to the location of the obstacle. It could also mean that most users were more comfortable with their guesses that they did not feel the urge to change them significantly. Although some users still made poor guesses as indicated by the values of $e > 6m$. The MSE is 6.5883, which is lower than either Test 1A or 1B for the image-based case. Although the Test 2 map had viewer landmarks, appeared to have performed better at identifying obstacles. One reason for this is that there were fewer distance intervals at the maximum distance was at $d = 20.74m$, which is lower than the maximum distances for Test 1A and 1B (27.15m and 28.67m respectively).

Regardless of differences in error values, the same overall trend still holds - that the farther the distance, the more prone to error the user will be in determining obstacle position on the map.

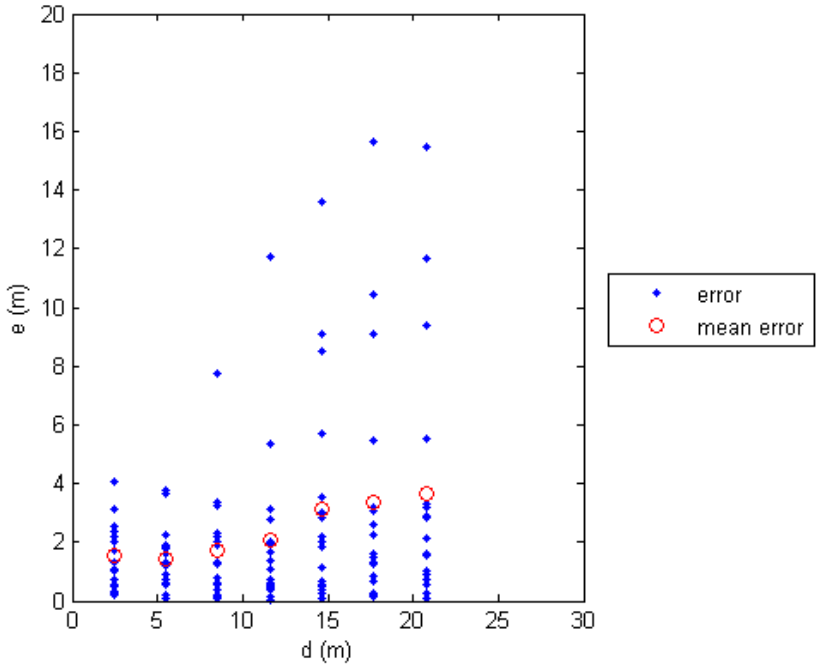


Figure 6.5: Error over distance for image-based Test 2.

Figure 6.6 shows the error over the distance using all of the image data combined. Some users were able to get a good idea of the obstacle's position even at large distances. However, most users were only able to determine the actual position of the obstacle at closer distances. The maximum mean error is 6.22m at $d = 21.05$, which belongs to Test 1A.

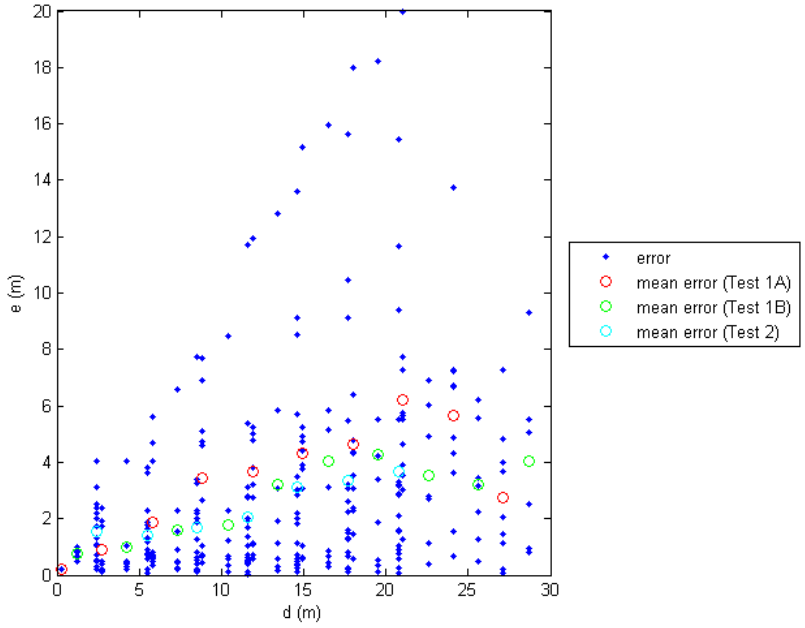


Figure 6.6: Error over distance for all image-based tests.

6.2.2 Video Test Results

Figure 6.7 shows the plot of error over distance for video-based Test 1A. The data is relatively sparse compared to its image-based counterpart. Most participants avoided placing obstacles on the map until until the robot was at a certain distance from the obstacle. Also, because of the inherent time constraint, each participant did not have enough time to view each video frame and determine the obstacle's distance. The points appear more scattered than the image-based case. However, there appears to be a relatively high concentration of points below the $e = 4$ line at $d < 10$, which would indicate that most users were more comfortable placing points when the distance between the robot and the reference obstacle was less than 10m.

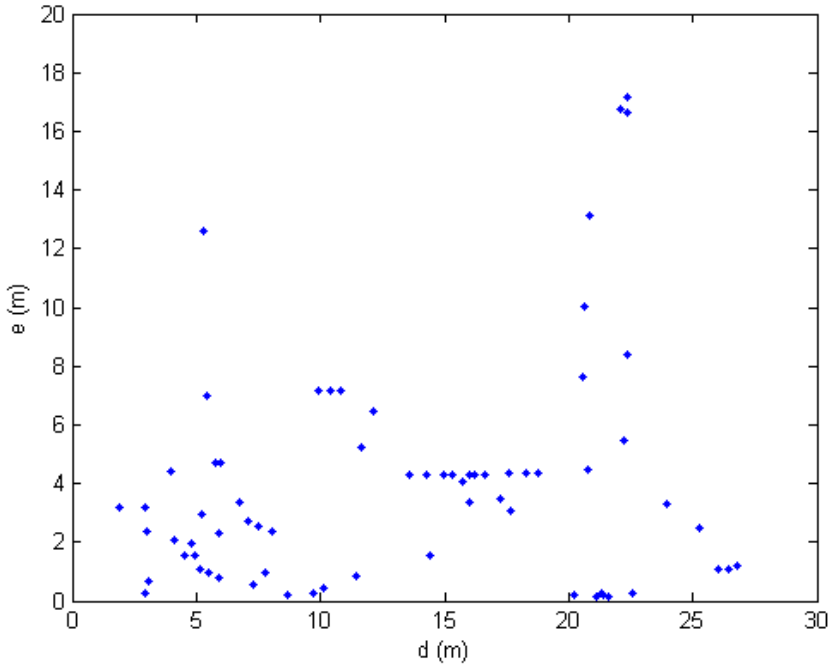


Figure 6.7: Error over distance for video-based Test 1A.

Figure 6.8 shows the plot of error over distance for video-based Test 1B. Here, the points are even more sparse compared to Test 1A. Some issues were encountered with setting up the test, such as the network failing in certain areas causing the robot to navigate incorrectly. The robot did not always reach the goal. Also, because of the aforementioned issues, the users ended up placing fewer obstacles on the map. However, the plot shows most points to be in the lower distane range. The greatest number of points is concentrated at $d < 15m$.

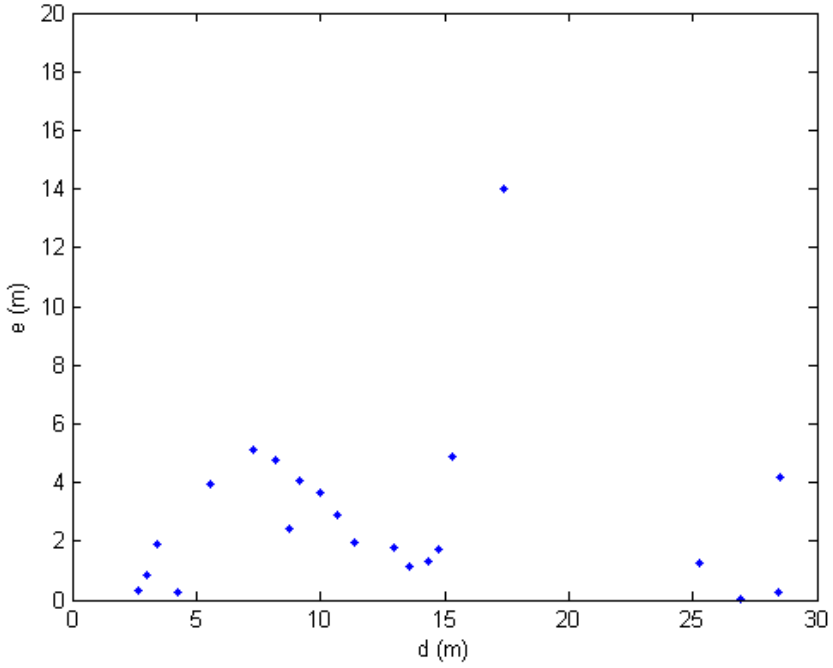


Figure 6.8: Error over distance for video-based Test 1B.

Figure 6.9 shows a denser set of points than video-based Test 1A and 1B. This particular test had more participants than either Test 1A or 1B. Due to complications with Test 1, some participants were asked to do only Test 2. The plot shows a high cluster of points at $d < 5m$, which progressively spreads as d becomes larger. However, most points still lie below $e = 4m$. Between $e = 2m$ and $e = 4m$. The data could have been improved

if additional instruction was given to all participants to keep adding obstacles as the robot was moving, establishing multiple best guesses at several distances instead of just one or two. This instruction was only given to the latter set of participants, who were told to put an obstacle at least every second as the robot was moving toward to goal. Interestingly, there appears to be a horizontal line of points. This is caused by one of the latter participants adding an obstacle multiple times on a singular location on the map, regardless of the distance.

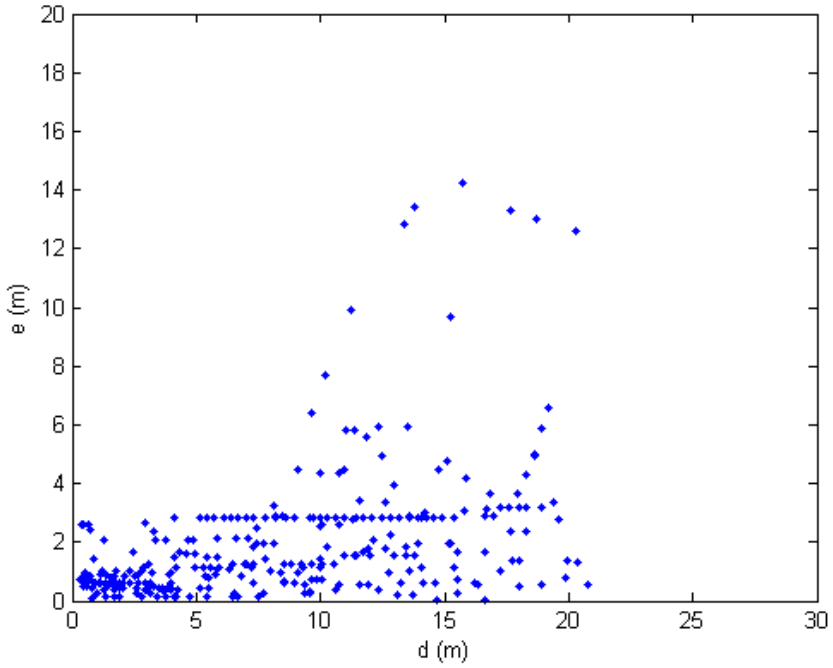


Figure 6.9: Error over distance for video-based Test 2.

Figure 6.10 shows the error over distance plot using all the data in the video-based tests. The plot closely resembles Figure 6.9 because the latter contributed more data.

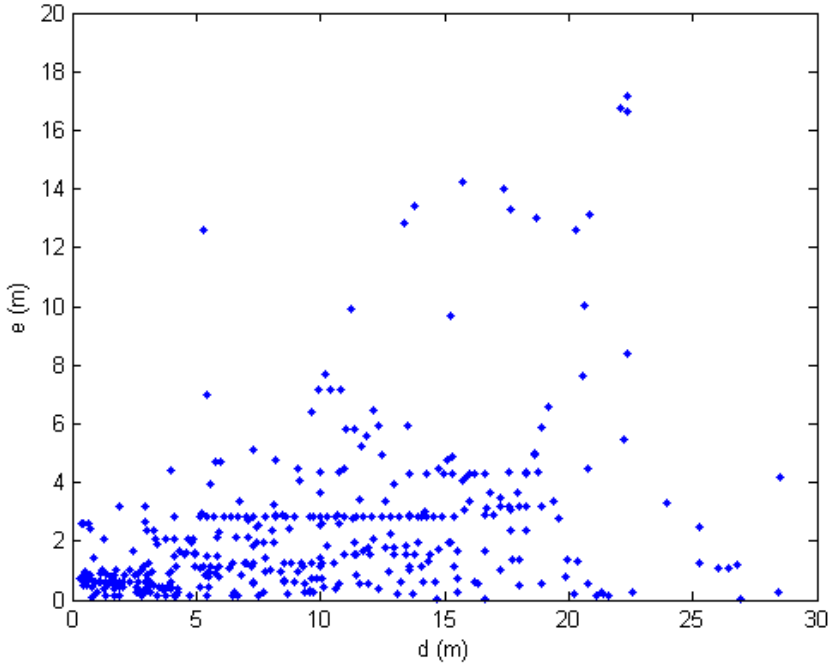


Figure 6.10: Error over distance for all video-based tests.

6.2.3 Image Versus Video Results

Figure 6.11 shows all image and video data points in one plot. The image data points are more spread out across the *d* range, which is expected. However, the video data points are concentrated towards the lower *d* range. Since users were given unlimited time to look at images in the image test, they would be able to identify landmarks on the video and match them to similar landmarks on the map and develop a better idea of the position of the obstacle. This results in users having lower *e* values even at large distances. In contrast, the video test had an implicit time constraint on the users. Since the robot was moving, the user only had a limited time to focus on landmarks in each image frame. The scene was continuously changing. Most users decided to take their time and only plot when they had a good idea of where the obstacle was in the map. This resulted in more points falling

in the $d < 5m$ range. There were still several outliers, however, which would indicate that some users were able to get a lower error even at large distances, which could be due to familiarity with environment or the test from the results of other participants.

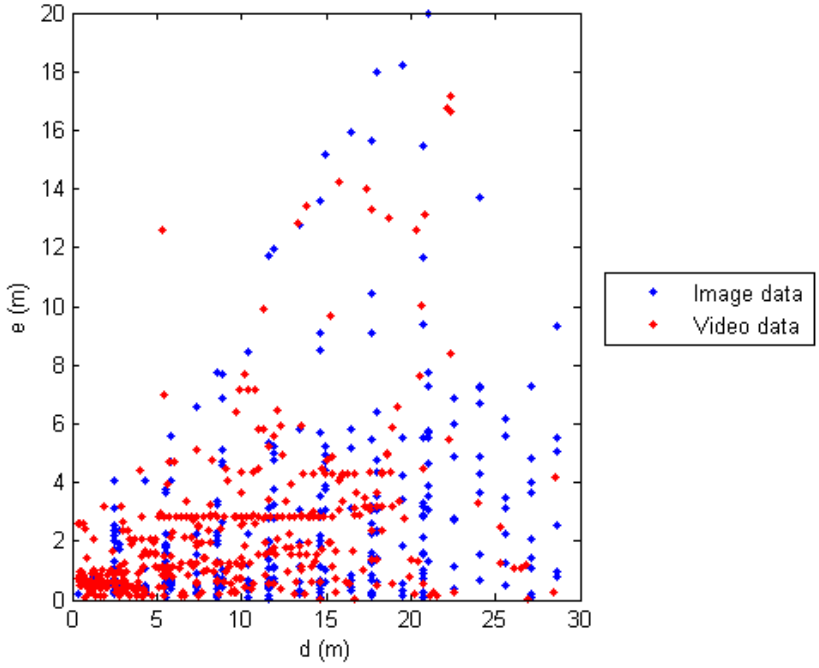


Figure 6.11: Image versus video data.

6.2.4 User Survey

The results of the survey showed that the participants were a relatively homogeneous group. Most of them were familiar with the test area, and played video games. Also, there ages were similar as well as their areas of study (i.e. they were all students in either computer science or computer engineering. Therefore, no distinction could have been established regarding performance based on background.

CHAPTER 7: CONCLUSION

This work presented a method of enhancing mobile robot navigation by allowing a human operator to update the robot's map in real-time using a remote navigation interface. The interface enabled a user to add obstacles on the map to represent an unmapped obstacle that the robot may or may not sense with its sensors. The method could be used to guide the movements of multiple robots that have intersecting paths. For instance, if a particular area is blocked and deemed impassable, the user can simple add obstacles to the area to indicate that it cannot be traversed. Then, each robot would update its map and recalculate a new path around the blocked area.

Experiments show the validity of this method. Robot navigation improved due to obstacle placement on a map with an unmapped obstacle. The robot was able to plan a path ahead of time to avoid an obstacle instead of relying on its sensors for obstacle avoidance once it is near the obstacle. Also, experiments conducted on human operators show that the greater the distance of the robot from an obstacle, the more difficult it is to judge to obstacle's position on the map. However, given enough time in observing the obstacle from a distance, the user may be able to make use of any landmarks on the map for estimating the position of the obstacle. The more landmarks that are present in the robot's map and video data, the easier it is for the user to determine to obstacle's position.

This work could be further improved in several ways. First, in order to improve the results of the user perception test, a more diverse group of participants is needed. The

current work used a relatively homogeneous group of participants having relatively the same background and experience. Second, the image-based test could be timed so that user must place an obstacle after a certain time interval. This would be closer to the expected scenario, which requires that the robot be in continuous motion toward a goal. Third, the RNI application could be improved by integrating either a video stream or still images as a separate window. This would make setup of the test much easier since only one application is required to handle both the map and the video. Further development could be done on the navigation interface by applying more human-robot interaction (HRI) principles to enhance its usability.

One long-term goal that should be pursued is that of creating a navigation interface that allows for control of multiple robots. The interface should allow various levels of control of the mobile robots. In this way, the capability of a human user to control multiple robots using the proposed method can be tested.

Additional long term enhancements would include adding click-and-drag functionality to alter the robot's path, use of more advanced path planning algorithms that support dynamic map changes without needing to halt, integrated mapping using a laser range finder or stereo-vision camera, and finally, updating the navigation interface for commercial or industrial use.

LIST OF REFERENCES

[1] X. Dai, H. Zhang, and Y. Shi, "Autonomous Navigation for Wheeled Mobile Robots-A Survey," in *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*. IEEE, Sept. 2007, pp. 551–551.

[2] M. E. Dempsey, "U.S. Army Unmanned Aircraft Systems Roadmap 2010-2035," 2010.

[3] S. Y.-S. Lee, "An augmented reality interface for multi-robot tele-operation and control," Ph.D., Wayne State University, United States – Michigan, 2011.

[4] J. Y. C. Chen, M. J. Barnes, and M. Harper-sciarini, "Supervisory Control of Multiple Robots : Human-Performance Issues and User-Interface Design," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 4, pp. 435–454, 2011.

[5] R. Kala, A. Shukla, R. Tiwari, S. Rungta, and R. Janghel, "Mobile Robot Navigation Control in Moving Obstacle Environment Using Genetic Algorithm, Artificial Neural Networks and A* Algorithm," in *2009 WRI World Congress on Computer Science and Information Engineering*. IEEE, 2009, pp. 705–713.

[6] S. Garrido, L. Moreno, D. Blanco, and M. L. Munoz, "Sensor-based global planning for mobile robot navigation," *Robotica*, vol. 25, no. 02, pp. 189–199, Mar. 2007.

[7] S. H. Cho and S. Hong, "Map Based Indoor Robot Navigation and Localization Using Laser Range finder," *2010 11th International Conference on Control Automation Robotics & Vision*, no. December, pp. 7–10, Dec. 2010.

[8] O. Ehtemam-Haghighi, "Autonomous navigation algorithms for indoor mobile robots," M.Sc.Eng., Lakehead University (Canada), Canada, 2009.

[9] D. Santosh, S. Achar, and C. V. Jawahar, "Autonomous image-based exploration for mobile robot navigation," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, May 2008, pp. 2717–2722.

[10] D. Correa, D. Sciotti, M. Prado, D. Sales, D. Wolf, and F. Osorio, "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on*, 2012.

[11] A. Tsalatsanis, "Control of autonomous robot teams in industrial applications," Ph.D., University of South Florida, United States – Florida, 2008.

[12] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. â. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, Aug. 2008.

[13] L. Doitsidis, A. Nelson, K. Valavanis, M. Long, and R. Murphy, "Experimental validation of a MATLAB based control architecture for multiple robot outdoor navigation," in *Proceedings of the 2005 IEEE International Symposium on, Mediterranean Conference on Control and Automation Intelligent Control, 2005.* IEEE, 2005, pp. 1499–1505.

[14] R. R. Murphy, "Validation of a Distributed Field Robot Architecture Integrated with a MATLAB-Based Control Theoretic Environment," *IEEE Robot. Autom. Mag*, vol. 13.3, no. September, pp. 93–107, 2006.

[15] M. T. Long, "Creating a distributed field robot architecture for multiple robots," Master's thesis, University of South Florida, 2004.

[16] N. Shiroma, N. Sato, Y. Chiu, and F. Matsuno, "Study on effective camera images for mobile robot teleoperation," in *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No.04TH8759).* IEEE, 2004, pp. 107–112.

[17] S. K. Cho, H. Z. Jin, J. M. Lee, and B. Yao, "Teleoperation of a Mobile Robot Using a Force-Reflection Joystick With Sensing Mechanism of Rotating Magnetic Field," *IEEE/ASME Transactions on Mechatronics*, vol. 15, no. 1, pp. 17–26, Feb. 2010.

[18] O. Martinez-Palafox and M. Spong, "Bilateral teleoperation of a wheeled mobile robot over delayed communication network," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 3298–3303.

[19] N. Mitsou, S. Velanas, and C. Tzafestas, "Visuo-Haptic Interface for Teleoperation of Mobile Robot Exploration Tasks," in *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, Sept. 2006, pp. 157–163.

[20] C. W. Nielsen, M. A. Goodrich, S. Member, and R. W. Ricks, "Ecological Interfaces for Improving Mobile Robot Teleoperation," vol. 23, no. 5, pp. 927–941, 2007.

[21] A. Kelly, N. Chan, H. Herman, D. Huber, R. Meyers, P. Rander, R. Warner, J. Ziglar, and E. Capstick, "Real-time photorealistic virtualized reality interface for remote mobile robot control," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 384–404, Oct. 2010.

[22] D. Sanders, "Analysis of the effects of time delays on the teleoperation of a mobile robot in various modes of operation," *The Industrial Robot*, vol. 36, no. 6, pp. 570–584, 2009.

[23] J. A. Gomer, "Spatial perception and robot operation: The relationship between visual spatial ability and performance under direct line of sight and teleoperation," Ph.D., Clemson University, United States – South Carolina, 2010.

[24] L. Upham Ellis, "Perception and displays for teleoperated robots," Ph.D., University of Central Florida, United States – Florida, 2008.

[25] B. Doroodgar, M. Ficocelli, B. Mobedi, and G. Nejat, "The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots," *2010 IEEE International Conference on Robotics and Automation*, pp. 2858–2863, May 2010.

[26] J. Koch, M. Reichardt, and K. Berns, "Universal web interfaces for robot control frameworks," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2008, pp. 2336–2341.

[27] M. Kadavasal Sivaraman, "Virtual reality based multi-modal teleoperation using mixed autonomy," Ph.D., Iowa State University, United States – Iowa, 2009.

[28] P. Nebot and E. Cervera, "Experiences in HRI: Design of an interface for a team of multiple heterogeneous robots," *2009 2nd Conference on Human System Interactions*, pp. 284–287, May 2009.

[29] H. Tang, X. Cao, A. Song, Y. Guo, and J. Bao, "Human-robot collaborative teleoperation system for semi-autonomous reconnaissance robot," in *2009 International Conference on Mechatronics and Automation*. Ieee, Aug. 2009, pp. 1934–1939.

[30] R. A. Chadwick, "Operating Multiple Semi-Autonomous Robots: Monitoring, Responding, Detecting," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 50, no. 3, pp. 329–333, Oct. 2006.

[31] N. Sato, K. Kon, and F. Matsuno, "Navigation interface for multiple autonomous mobile robots with grouping function," *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 32–37, Nov. 2011.

[32] B. Keyes and H. A. Yanco, "LASSOing HRI : Analyzing Situation Awareness in Map-Centric and Video-Centric Interfaces," pp. 279–286, 2007.

[33] M. Lewis and J. Wang, "Human control for cooperating robot teams," *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, 2007.

[34] E. V. Cross, "Human coordination of robot teams: An empirical study of multimodal interface design," Ph.D., Auburn University, United States – Alabama, 2009.

[35] A. Correa, M. R. Walter, L. Fletcher, J. Glass, S. Teller, and R. Davis, "Multimodal interaction with an autonomous forklift," pp. 243–250, Mar. 2010.

[36] S. Kim, J.-h. Park, and S.-k. Park, "Vision-based Human Augmented Mapping for Indoor Environments," 2009.

[37] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, 2006.

[38] K. M. Passino, *Biomimicry for Optimization, Control, and Automation*. Springer, 2004.

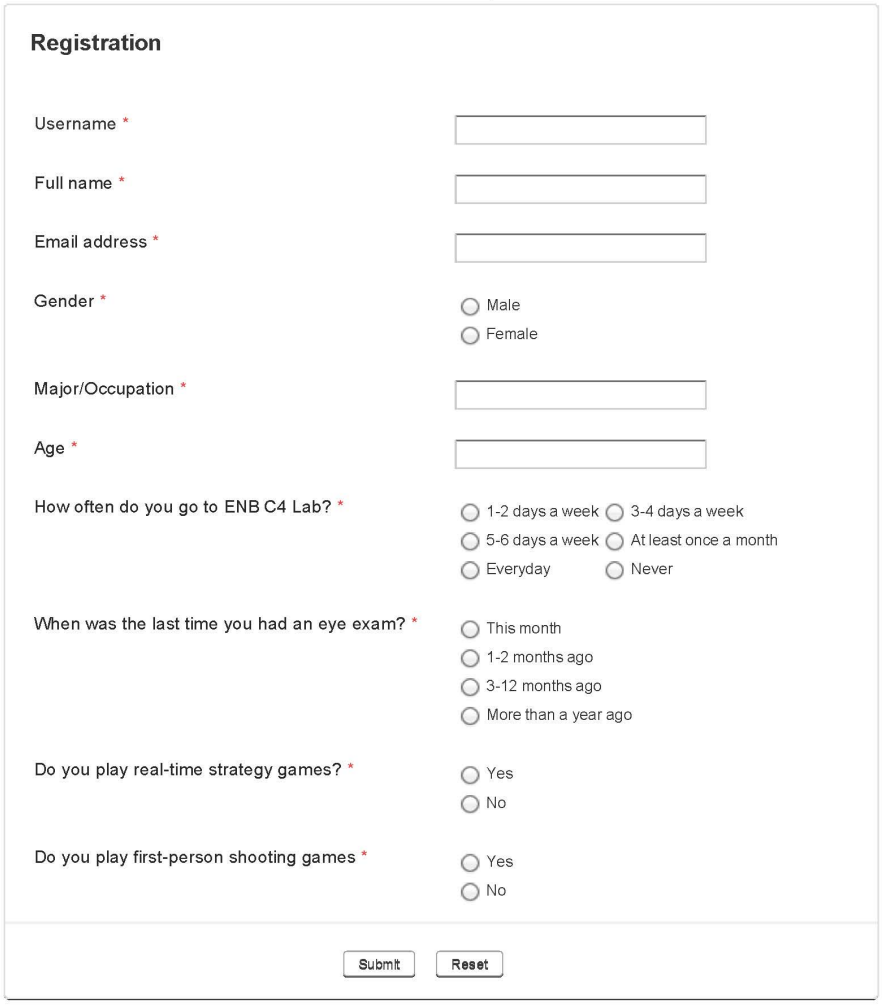
[39] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[40] B. P. Gerkey, R. T. Vaughan, and A. Howard, "In Proceedings of the International Conference on Advanced Robotics (ICAR 2003) The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems," no. Icar, pp. 317–323, 2003.

[41] D. J. Barrett, R. E. Silverman, and R. G. Byrnes, *SSH, The Secure Shell: The Definitive Guide*, second edition ed. O'Reilly Media, May 2005.

APPENDICES

Appendix A: User Perception Test Survey Form



Registration

Username *

Full name *

Email address *

Gender *
- Male
- Female

Major/Occupation *

Age *

How often do you go to ENB C4 Lab? *
- 1-2 days a week
- 3-4 days a week
- 5-6 days a week
- At least once a month
- Everyday
- Never

When was the last time you had an eye exam? *
- This month
- 1-2 months ago
- 3-12 months ago
- More than a year ago

Do you play real-time strategy games? *
- Yes
- No

Do you play first-person shooting games *
- Yes
- No

Submit Reset

1/1

ABOUT THE AUTHOR

Carlos Ezequiel earned his bachelor's degrees in Physics and Computer Engineering from the Ateneo de Manila University, Philippines in 2006 and 2007 respectively. He was awarded a Fulbright scholarship grant to pursue a Master's degree in Computer Science at the University of South Florida. He had worked as a software engineer specializing in embedded systems development, and later, in server applications development. He is one of the founders of Skyeye, Inc., a Philippine-based startup company involved in UAV decision support systems. His research interests include autonomous unmanned systems, bio-inspired robotics, computer vision and physics.