ROBOTIC TOUR GUIDE PLATFORM

By

Jesse Fish

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Advisor: Wyatt Newman

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

January, 2013

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Jesse Fish

candidate for the Master of Science degree *.

(signed) Wyatt Newman
(chair of the committee)

Francis Merat

Greg Lee

(date) 7-10-2012

*We also certify that written approval has been obtained for any proprietary material contained therein.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AMCL: Augmented Monte Carlo Localization

LIDAR: LIght Detection And Ranging

PC: Personal Computer

ROS: Robot Operating System

SLAM: Simultaneous Localization and Mapping

GLOSSARY

A Priori Map: A map of an environment that a robot can use giving information about the environment before any sensor readings are collected.

cRIO: Compact RIO, a reconfigurable embedded control and acquisition system made by National Instruments

Holonomic: A robot is considered to be holonomic if the number of controllable degrees of freedom is equal to the total degrees of freedom for the platform. An automobile is an example of a non-holonomic vehicle.

LIDAR: A system that uses ultraviolet, visible, or near infrared light to image objects.

Teleoperation: Operation of a machine from a distance.

Acknowledgements

First, I would like to thank my thesis advisor, Dr. Wyatt Newman. His guidance and wisdom helped immensely throughout this thesis. I would like to thank Larry Sears as well. Without his great generosity and vision this thesis would not have been possible. I would also like to thank Ed Burwell and George Daher for their help and expert advice. Eric Perko, Chad Rocky, and Toby Waite were all extremely helpful throughout this thesis and I would like to extend my thanks to each of them.

On a personal note I would like to thank my parents for their continuous support and motivation. And I would like to thank Beth Murray whose constant support and guidance helped me through my masters program.

Robotic Tour Guide Platform

Abstract

By

JESSE FISH

This thesis describes the development of a mobile robot research platform (Roberto) designed for use in developing LIDAR, vision, and other sensor-based mapping and localization systems, path planning and steering systems, and emotive human-robot interaction systems. The platform has been designed to allow for flexibility and extensibility while assuring safety and stability. The ROS open source library has been utilized to leverage middleware solutions common to many problems in robotics. This platform has been demonstrated to have the capability to operate safely around humans in a dynamic and unmodified environment, giving tours of the Glennan building at Case Western Reserve University. This platform allows for teleoperation as well as full autonomous functionality.

Chapter 1: Introduction

This thesis describes the development of Roberto, an electric wheelchair-based mobile robot developed to function as a tour guide and research platform at Case Western Reserve University. Roberto needed to meet the needs of graduate and undergraduate researchers in the fields of artificial intelligence, computer vision, control systems, and human computer interaction. Additionally it needed to safely give tours of the Glennan EECS building at Case Western Reserve University. This thesis will examine previous successful mobile robotics platforms, identify effective design choices, and give a detailed overview the design and systems of Roberto.

1.1 Previous Successful Platforms

The first robotic tour guide was Rhino, which was installed in Deutsches Museum Bonn (Germany) in mid-1997 [40]. Rhino used a holonomic movement base with a circle of sonar sensors interpreted by a neural network along with a stereo camera system to map its environment in an occupancy grid. It also used a powerful probabilistic sensor fusion technique for global localization [41]. Global localization is a technique to localize the robot while considering all possible locations in an environment [41].

Rhino's success motivated the creation of a second tour-guide robot named Chips (or Sage), another holonomic robot which localized off of bright optical markers [24]. In the case of Chips, the effectiveness of the platform was judged on

the robot being accessible, educational, entertaining, and appealing to a broad range of visitors. Chips was very successful in reaching the goals set for it: to make a simple robot that could operate a lifetime without human intervention [23]. There were drawbacks to this approach, however. Chips used a single-threaded architecture, which limited its ability to handle much processing between control loops. Also, in order for it to localize, its environment needed to be altered [23].

Minerva successfully toured the Smithsonian for 2 weeks in 1998, interacting with thousands of visitors while informing and entertaining them. Minerva utilized a generic software approach for both navigation and human robot interaction. This software allowed Minerva to: navigate in dynamic environments, navigate in unmodified environments, engage in short-term human-robot interaction, and have a virtual tele-presence. Minerva used mosaic matching vision-based localization, had a robotic face that could display various emotions, and used genetic algorithms to continuously learn over time. It also used a holonomic movement base. Minerva was superior to Rhino and Chips in a number of ways. It learned its maps and used ceiling mosaics for localization. Its path planner took uncertainty into account to avoid planning through featureless space. It also had a much richer interactive experience, including emotional states the robot could use to communicate intents [40].

In 2001 Rice University created an outdoor tour-guide robot named Virgil. This rugged mini-truck based (non-holonomic) platform had great success in outdoor localization, fusing GPS with wheel- and gyro-based odometry using Extended Kalman Filters [39]. The project was directed at tackling the challenges of

outdoor navigation without modifying the environment. A drawback of this focus was that the robot had minimal human interaction. However it was still successful despite this drawback, showing that it is not always necessary for a robotic tour guide to have a face or emotion system to capture peoples' attentions and imaginations.

RoboX was a successful tour-guide robot at the Swiss National Exhibitions in 2002. The team that made RoboX identified several important factors for a successful robotic tour guide. The robot must keep collision risk low and ensure the effects of a collision are harmless. Smooth motion was also important as visitors anticipate movement when they follow the guide. The obstacle avoidance control loop needed to run fast, not only in order to run in real-time, but also to allow for enough processing resources for localization, sensor acquisition, a web server, and motor control. RoboX used known obstacle avoidance and path planning algorithms. The drawbacks and advantages of these algorithms were weighed with the goal of consistent fusion in mind. Modifications were made to these algorithms to improve them without compromising functionality. A graph-based *a priori* map was used for global planner, and the local path planner used a combination of the Elastic band and NF1 [29] approaches. The platform was used to test the dynamic window approach for obstacle avoidance in a dynamic environment. It was able to successfully navigate and give tours smoothly through crowded exhibitions [29].

Many other robotic platforms have functioned successfully as tour guides, beyond the few listed here [14, 18, 25, 43]. The algorithms, hardware, and software

architectures varied widely across all of these robots. All of these successful robots show that there is no one best solution for a robotic tour guide.

1.2 Leveraging of Existing Solutions

Many of the problems in mobile robotics have been solved with varying degrees of success. Algorithms exist that can approach the problems of steering, path planning, localization, and world modeling [2, 6, 7, 10, 11, 13, 20, 21, 37]. The purpose of this thesis is not to improve on these algorithms, but to apply them to a purpose--creating a stable and reliable platform that can be used as a tour-guide for the university and to test and improve new algorithms in the field of robotics.

The problem of implementing these algorithms and integrating them into a cohesive architecture is one nearly all robotics researchers have faced. Developing software for mobile robot applications is a tedious and error-prone task [43]. Because this is work that frequently needs to be repeated over and over again, researchers have worked on developing a universal software architecture for mobile robots. Over the years many architectures have been created for this purpose [1, 3, 8, 15, 17, 30, 32, 33, 34, 35, 38, 43, 44]. Several of these architectures were developed under open-source licenses. These codebases can be leveraged to speed up development and to improve the performance and stability of a new mobile robot platform. Miro, ROS, Open RDK, and many more of these projects were developed with the intention of being used as a middleware solution for mobile robots 18, 38, 43, 49]. These projects also sought to establish communities to develop and maintain these open-source robotics platforms.

Throughout the development of Roberto, the leveraging of existing solutions was always preferred to reinventing a solution. This reuse led to a better, more stable, and easier to use platform. Using an open codebase reviewed by thousands of people is a much better design decision than using a closed codebase reviewed by a handful of researchers. There are times when an existing solution is not available or the existing solution is not well suited to the problem. In these cases the new code should be tested thoroughly and if possible integrated into an open framework for others to use and test. Testing, code coverage, and reliability are all needlessly limited by a non-collaborative approach.

1.3 Design Goals

Before planning and building this research platform it was important to set goals to guide its design. These goals were picked based on previous successful hardware and software architectures' design goals. Portability, ease of use, modularity, robust design, safety, and stability were chosen as the design goals for Roberto.

Portability and ease of use are very important for this platform. In 1994, John Fox stated that, in robotics, programs should be *portable* from system to system and that interfacing with sensors should be uncomplicated and *easily implemented* [22]. Roberto will be used by undergraduate and graduate researchers whose work should not be impeded by a difficult to use platform. These researchers also may need to add sensors to the platform as well as program new modules. Streamlining this integration process serves to make Roberto a better platform. The programs written for Roberto also should

be portable to other systems. Because Roberto multiple researchers may wish to use a system like Roberto at conflicting times, Roberto's architecture should be compatible with several of the systems in the lab.

There are numerous positive benefits of modular design in both hardware and software systems in robotics. It allows for incremental design of a system, being able to build one part at a time without needing to completely overhaul the system. In software, it enables code reuse. In most applications it is desirable to design a system that integrates modular libraries that can be easily reused [1, 19]. Modularity also allows for fast reconfiguring of the platform for different applications [41]. This is a desirable feature for Roberto; as it will be used in many different research applications, it is important to minimize the effort and time spent configuring the platform for a desired task. Modularity is a common cornerstone of robot design and is central to the design of Roberto [4, 15, 22, 38, 40, 41].

The safety and stability of a robot platform are the most important issues to consider during design. A robot can be difficult to work with, hard to develop on, and completely overspecialized, so long as the safety of all people interacting with it is assured. To guarantee a system is safe, all failure modes must be known and there must be a failsafe response for each failure mode. Stability is somewhat coupled with safety. The more stable a robot is, the fewer catastrophic failure modes it will have. Roberto is a robot that will be representing the face of the university for prospective students and their families. It will interact with hundreds of people including small children. Consistently in the design of Roberto, safety and stability were prioritized above all else.

1.4 Organization of This Thesis

This thesis details Roberto's design and system structure. Chapter 2 describes the hardware architecture of the robot starting with the physical design in 2.1, then covering the electrical systems in 2.2. The sensors and components selected for the platform are detailed in 2.3. The software architecture is covered in chapter 3. Here the software on the cRIO is discussed in 3.1. There were several options for what software to use on the Intel PC. These are detailed in 3.2. After deciding ultimately to use ROS, a detailed overview of the ROS system implemented on Roberto is covered in 3.3. Chapter 4 details various localization methods tested with the robot. In chapter 5 the steering and planning algorithms used on the platform are discussed. The human interaction systems on the robot are covered in chapter 6. Chapter 7 provides conclusions for this work, and chapter 8 details future work.

Chapter 2: Hardware Architecture

The Mobile Robotics Lab at Case Western Reserve University has built several electric wheelchair-based robots in the past. The most influential lab robots to this platform were: Jinx, Harlie, and ALEN. These robots have been used in the Intelligent Ground Vehicle Competition (IGVC)[46] and in student research.

Figure 2.0.1: Harlie, Jinx, and Roberto (left to right). Alen can be seen in the back right

When this work began, Roberto was a robot in disrepair that had been sitting unused in the lab for several years. Both the electronics and the physical design of Roberto were outdated and unusable for this thesis. The robot was completely disassembled to the wheelchair base and rebuilt back up from scratch. In this rebuilding process the three previous robots were analyzed to identify specific

design choices that benefited them. These designs were leveraged in the rebuilding of Roberto. Roberto's physical design and the decisions made leading to it are detailed in section 2.1. The electrical design is detailed in section 2.2 and the components and sensors used on the robot are discussed in section 2.3. Section 2.4 includes what software approaches were considered for Roberto and details the final software architecture.

2.1 Physical Design

The base of Roberto is a custom-built wheelchair base built and donated by the Invacare Corporation. It has 4 wheels: 2 large wheels powered independently by motors and 2 smaller passive casters. Because the powered wheels are independent of each other it can move forwards and backwards, turn in arcs, and turn in place.

The first and most noticeable difference between Roberto and the previous robots is that Roberto is much shorter. The lab's primary use for the previous robots has been outdoor mobile robotics competitions. GPS modules were installed on the robots for these competitions, as well as cameras. The GPS antennae work best when distanced from the EMI noise of the drive and computing electronics, and the camera's view was preferably high up. In Figure 2.0.1 (above) the GPS antennae can be seen at the top of Harlie and Jinx. However, building a tall robot has serious disadvantages. Primarily, it causes the robots to be top heavy. They can easily tip forward if they encounter sudden deceleration or even if they are simply travelling downhill. Because Roberto's intended environment is indoors, it has no need for a GPS module, nor for a high camera viewpoint. Because of this, when rebuilding the

frame, a conscious effort was made to make Roberto shorter. This not only makes it harder to tip the robot over, but also has the effect of making Roberto less intimidating to visitors. Some of the lab's larger robots, especially Harlie, which stands over 6 feet tall, scare people as they are travelling through buildings. From this observation and for stability purposes the decision was made to have Roberto be shorter.

Most of the lab's robots are built on electric wheelchair bases. The bodies of these robots are made of Bosch rail scaffolding with Plexiglas paneling. Bosch rail is light (aluminum), relatively cheap, and easy to machine if modifications are required. The lab also had plenty of it. Plexiglas, similarly, is easy to work with, cheap, and was in ample supply. Though wood and steel were also available, it was desirable to have Roberto match the basic look of the previous robots and be visually appealing to prospective students observing him. Because of this, Bosch rail and Plexiglas are what comprise Roberto's body.

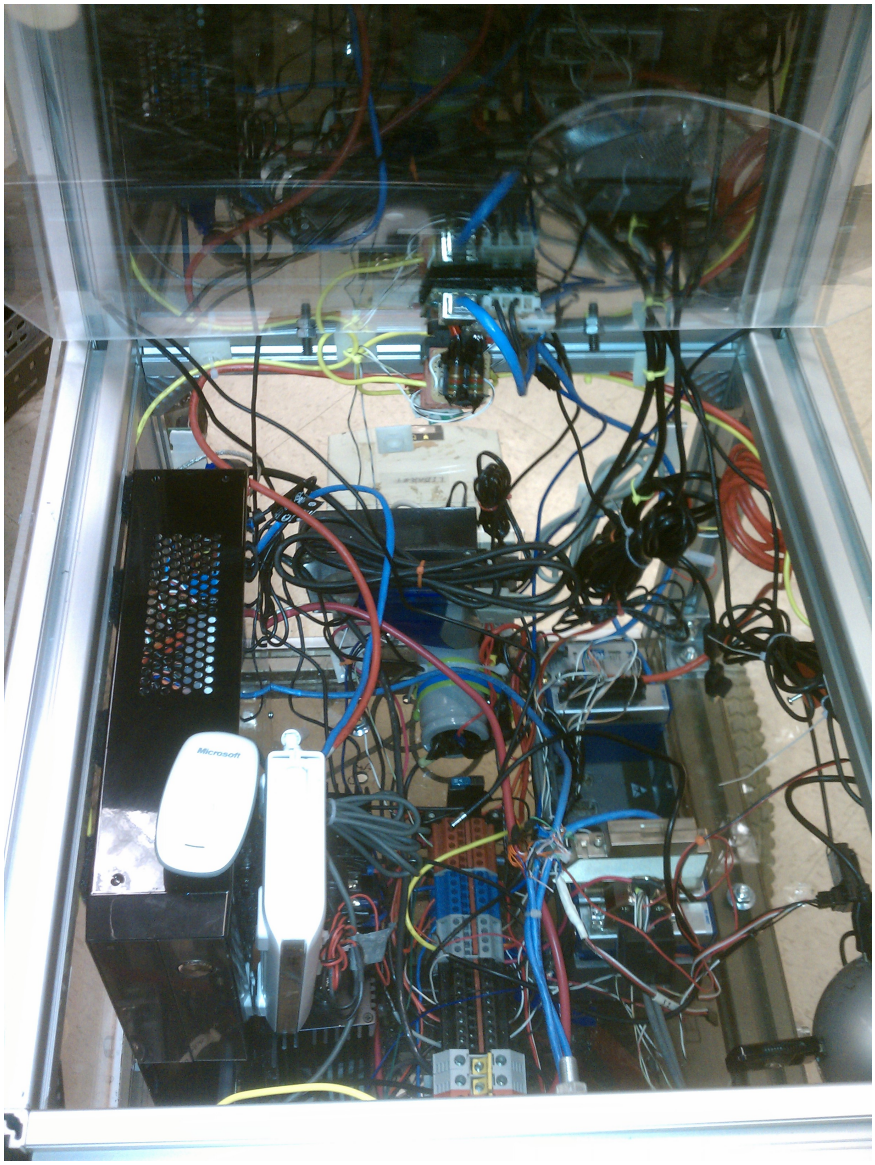

Figure 2.1.1: Roberto's internal component layout

The component layout within Roberto is a less obvious intentional design difference from the other robots in the lab. Jinx is a robot that is primarily used for outdoor competition and testing. It has a 2-shelf design that on the one hand impedes wiring or modifying the components while the robot is assembled, but on the other allows for the robot to be rapidly disassembled to make adjustments. Jinx's components are also difficult to see, both because they are stacked in two

layers and also because some of the Plexiglas forming its body is opaque. It was desired that Roberto's insides to be visible to all observers. Roberto therefore only used fully transparent Plexiglas panels. This constraint also led to a new design for the robot base--a single lidded box for all components (Figure 2.1.1). The majority of the components were affixed to the bottom of the box, with a few mounted to the interior sides and top. People receiving tours from the robot can look in the top or sides of the robot and easily see all the parts wired together. Most engineers and prospective engineers want to know how things work. Because Roberto is being used as a recruiting tool, it was important to make all of its pieces completely visible.

The previous robots have been equipped with small touch screen monitors, used mainly for debugging and live feedback for development. Because Roberto would be a tour guide it was decided that a larger monitor would be more engaging for displaying information. The small touch screen was kept from the previous designs with the intent of having the tour audience interact with the robot through it to get more information about the building and to create a more dynamic tour. At the university, tour guides usually give tours walking backwards. This way they engage the audience while they travel through the campus. It is for this reason that the monitors are facing backwards on the robot, to engage the audience while they walk behind it.

Safety was a major concern in the design of the robot. It is going to be interacting with people on a regular basis and used by students on future projects. These robots can be very dangerous, especially with inexperienced engineers. For

this reason there are multiple electronic emergency stop (E-stop) switches on the robot, and a wireless E-stop onboard as well. They have been placed so that anyone can quickly and easy press the large buttons and shut off the robot's motors completely (figure 2.1.2).

Figure 2.1.2: E-stop switch on Roberto with solenoid below

The previous robots in the lab (Jinx and Harlie) are built on newer wheelchair bases than Roberto. They each have an integrated battery housing, where each battery is placed in a plastic box, which can slide along a track below the robot. This allows for rapid removal and replacement of batteries, so a spare charged set can be swapped in at a moment’s notice. Roberto's battery holder was built with the wheelchair base. But it is nothing more than a large bolt and some

steel bent and welded together to hold batteries between the wheels. When this work started, the battery holder was heavily rusted, unsightly, and not secured to the robot safely. The holder since has been sandblasted, cleaned, and painted black. Nuts were also affixed to it to ensure it is safely secured to the robot. It still has a shortcoming over the other robots in that removing the batteries from Roberto is a time consuming task.

Figure 2.1.3: Battery Holder on Roberto

Finally, the laying out and affixing of the components of the robot was completed with the knowledge that it would be regularly used and updated over time. Many light components are secured with heavy duty Velcro to allow for easy removal and replacement with parts that may be of differing size and shape, though there are some longer-term components that are bolted into the Plexiglas or affixed to the Bosch frame.

2.2 Electrical Design

The Electrical design of the robot borrowed heavily from the design of previous robots in the lab. The system is powered by two 12-volt car batteries connected in series. The main voltage of the device is roughly 24 volts (as the battery charge changes this voltage can vary). All of the lab's wheelchair bases have a unique charging system. They use a 3-pin DMX connector that plugs into a 24-volt battery charger (an Invacare connector choice). This is odd because a 3-pin DMX plug is an audio plug usually used for microphone or speaker systems. There is a connector on the back of Roberto that accepts this interface for charging (Figure 2.2.1). Some of the robots in the lab use wheelchairs that have integrated charging and wiring into their design. This was not the case with Roberto. It was necessary to manually install the charging outlet and to wire the batteries into it.

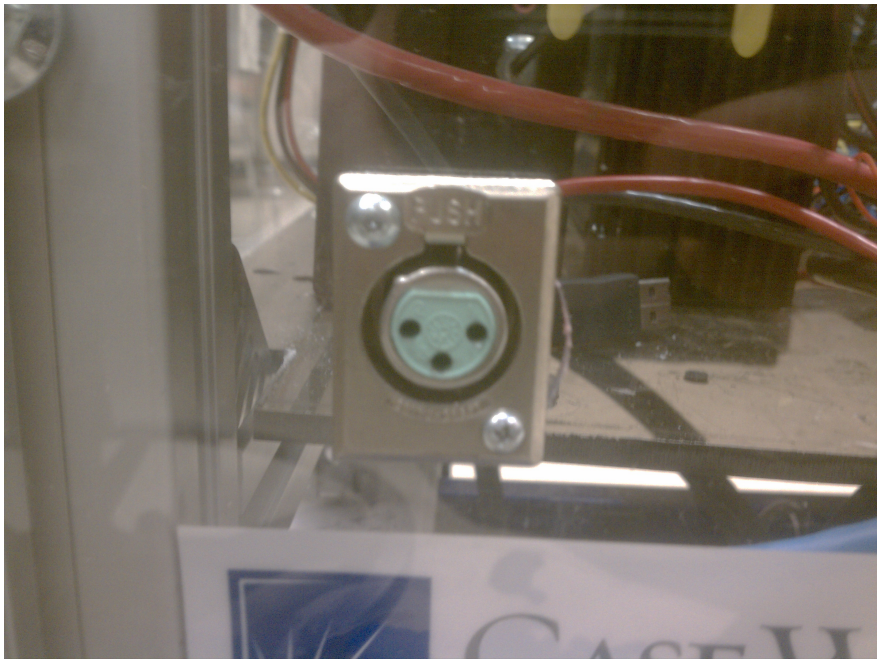

Figure 2.2.1: 3-pin DMX charging port on Roberto

The robot has a large bank of Phoenix rail in the bottom center of the base. Roberto originally used bus rails with screws for electrical connections. These were not insulated, unsafe, unsightly, and difficult to work with. Phoenix rail was chosen because it was readily available in the lab, it was used in the other robots with success, and it secures wires well with screws while insulating the connections with a plastic casing. On Harlie and Jinx there are many Phoenix rail segments mounted around the robots to serve as connectors as they are needed for various components. On Roberto all of the phoenix rail pieces are in one location: a central rail that runs down the middle of the robot. This made wiring cleaner and, by placing it in the center of the robot, parts could easily be placed around it to minimize wire run distances. On all of the robots the rails are color coded and grouped by the voltage they carry. This is important for the safety of the robot, the components housed in it, and the engineers working with it.

The robot provides power at 3 different voltages: 24v, 13.8v, and 5v. There is a power regulator that takes input from the 24v rail and drops it to 13.8, and then a smaller lab-made voltage divider that takes input from the 13.8v rail and drops it to 5V. These 3 voltages are consistent with the voltages on the other robots. This is important because the lab will regularly swap parts between robots to debug problems or repurpose parts. If the robots are not running at the same voltages then they cannot support this modularity of components. All power connections and wires are color-coded for these 3 voltages; red for 24v, blue for 13.8v, and grey for 5v. Black wires and connectors always mean ground, which is the negative terminal on the battery. Logic wires that carry communications signals will have different

colors depending on the voltage of the signal and the type of communication. The main 24v rail has an LED voltage meter attached to it. This displays the main line voltage on the back of the robot so engineers can see when the robot needs to be charged. There are also wires at each voltage level that feed into the cRIO to allow for software voltage measurements of each power level (fed through voltage dividers). This is useful for both error debugging and simply providing a health/status of the robot.

There are two other power regulators on the robot. One is a 24-volt dc-to-dc PC power supply. It was taken from the older robot, ALAN. It plugs directly into the motherboard of the onboard Intel PC. The other is a power supply for the LIDAR system. The LIDAR requires a clean input voltage signal. The lab has made its own power regulators for these sensors. Also, while there is not a power regulator, there is a large capacitor and resistor RC circuit, serving as a low-pass filter on the power going into the cRIO computer. The cRIO has a large operating voltage range, which is necessary because when the motors on the robot rapidly accelerate from a stand still or suddenly reverse direction, they create a spike in current drawn from the battery. This creates a drop in power going to the cRIO for a brief period of time. The RC circuit serves as a small reserve of power in these cases to ensure the cRIO does not reboot due to lack of power. Because this problem is present in all of the lab's robots, they all have a similar filter for the power of their respective cRIOs.

As stated before, safety is a primary concern in the design of this robot. All connections on the robot are insulated to minimize safety risks. The frame of the robot is also grounded to the negative terminal on the battery. This was not

previously the lab's standard practice. However, there was an accident where Jinx's GPS antenna was mounted to its frame and a wire inside the robot touched the frame as well. This caused a large spark and left burn marks where the antenna was conductively touching the frame. Since then, to protect electronics on the device and engineers using the device, the frame is grounded.

There are two different electrical shutdown systems on the robot. The first is a heavy-duty power switch that interrupts the positive terminal of the 24V battery from the Phoenix rail. When this switch is turned off, nothing on the robot has power. However, it can still be charged when this switch is off. All of the lab's robots have similar power switches. Roberto's is unique in that it is inside the robot rather than outside. It was placed inside the robot for two reasons. Firstly, when developing on Jinx the engineers frequently accidentally bumped the power switch, shutting down the robot while they were working on it. This was frustrating and increased development time. Secondly, because the robot will be around prospective students and their families, it was unwise to position the power switch to the robot within a child's (or childlike adult's) reach.

The second electrical shutdown system is a series of emergency stop (E-stop) switches that wire into a large solenoid (Figure 2.1.2). This solenoid interrupts the power from the 24V phoenix rail to the motor controller input. If any E-stops are triggered, the motors lose all power. There are two mechanical E-stops on the robot that have indicator LEDs signaling if they are activated or open. There is also a software E-stop that triggers a solid-state relay that breaks the E-stop wire. Lastly, there is a wireless E-stop that can interrupt this system and shut the motors down.

This is the most important E-stop because, generally, it is the first one that will be hit if the robot is misbehaving. This is because there should always be someone holding the wireless E-stop, ready to press it, while a tour is happening or any research is being done with the robot.

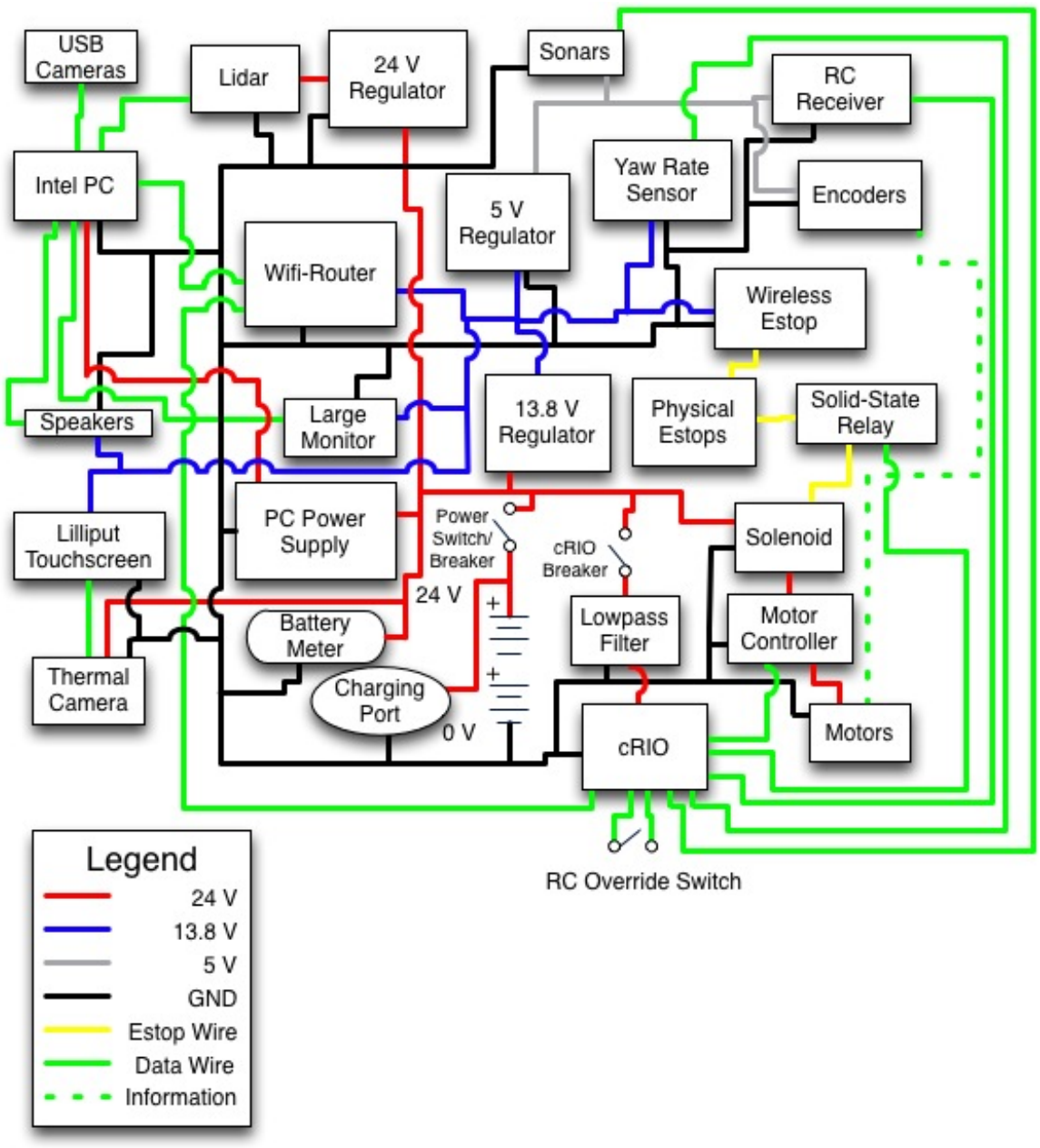

Figure 2.2.2: Circuit diagram of Roberto

Figure 2.2.2 shows a detailed circuit diagram of the robot. The average current drawn from the battery of the robot while stationary is 4 amps. The peak current drawn while moving is 50 amps. The average current while moving is 20 amps. The average battery life of the robot is 4 hours.

2. 3 Components and Sensors

The sensors and components on a robot dictate what it can detect and compute. The decisions of what to equip a robot with are influenced by what the robot should be capable of in known or unknown environments. Because Roberto will be used as a research platform for the next several years, it is impossible to predict every goal or task that it will face in the future. It has been designed to be a robust platform but primarily to be an indoor tour-guide robot. The modular design allows future researchers to add more sensors or components as they need them. The term "sensor" is used here to describe something that gives the robot information about its environment while "component" is a term for any modules that are used for computation, direct control, or other miscellaneous tasks. The main sensors on the robot are the wheel and motor encoders (Grayhill 61k Optical Encoders), yaw-rate sensor (Analog Devices ADXRS150 Gyro), LIDAR (SICK LMS291), sonar range finders (Maxbotix Ultrasonic Rangefinder - LV-EZ1), a thermal camera, a USB web camera (Logitech Webcam Pro 9000), and a Lilliput touch screen. The main components on the robot are the cRIO and its modules, Intel PC, Sabertooth 2x25 motor controller, wireless router, USB hub, XBox360 USB

controller wireless receiver, RC controller receiver (Xtra Speed 2.4 GHz radio system), and USB to serial converter.

2. 3 .1 Sensors

Wheel encoders are used to detect how much the wheels have moved over a small delta time. These sensors (Grayhill 61k Optical Encoders) are high-resolution optical encoders that give reasonable short-term change in position and rotation data for the robot's movement. However, the values of these sensors can have errors both due to wheel slip, caused by the wheel slipping across the ground, and due to calibration error. They are small rotational sensors that should rotate at a rate relative to the wheels. This is accomplished on Roberto by placing a gear on the encoder and the wheel axle and connecting the two with a toothed track. The sensor must be calibrated so the robot has a good measurement of the relationship between the angle of rotation on the encoder and the distance traveled by the wheel. Roberto originally had forward protruding spokes that the wheel encoders were mounted on. There was a toothed track that connected the encoders to the wheel axles. This design was both ugly and dangerous. One of the first things done when remaking Roberto was tucking these encoders into the underside of the robot and removing the spokes that people could easily walk into when moving around the robot (Figure 2.3.1).

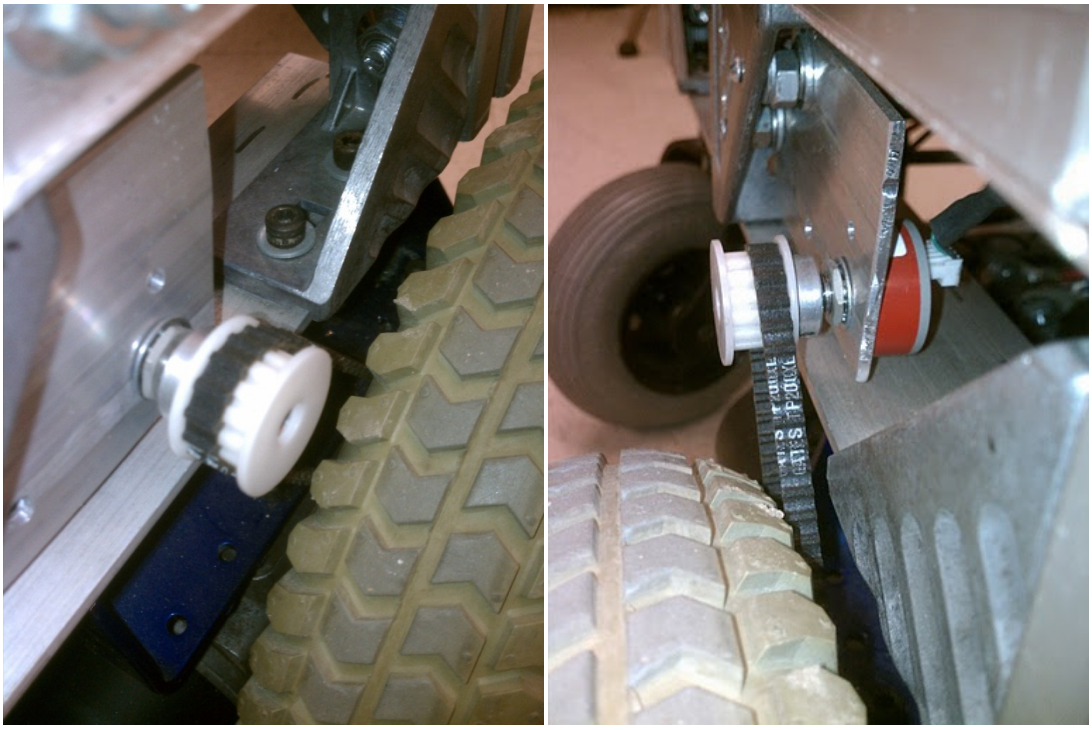

Figure 2.3.1: Wheel encoder tucked under the robot

The motors on Roberto are heavily geared, like most electric wheelchair motors. This means the major axle of the motor rotates at a much faster speed than that of the wheel axle. The motor encoders on Roberto are identical to the wheel encoders, but they attach to the rotor of the motor to give feedback information to the motor control system (a PID system). On Roberto the back of each motor was sealed with a steel plate and electric brakes that no longer functioned. The plates and brakes were removed and holes were drilled into the plates. A pair of mounts and couplings were designed and machined to attach the encoders to the axles (Figures 2.3.1 and 2.3.2).

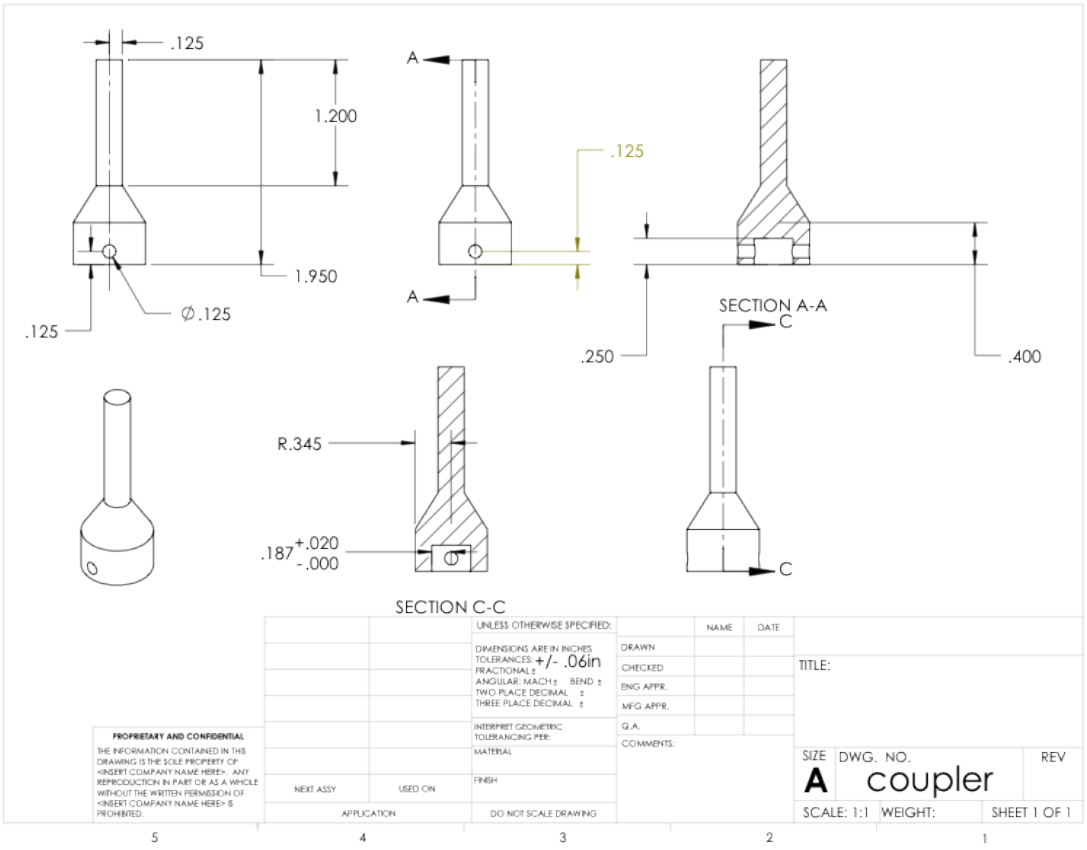

Figure 2.3.2: Schematic of motor-encoder coupler

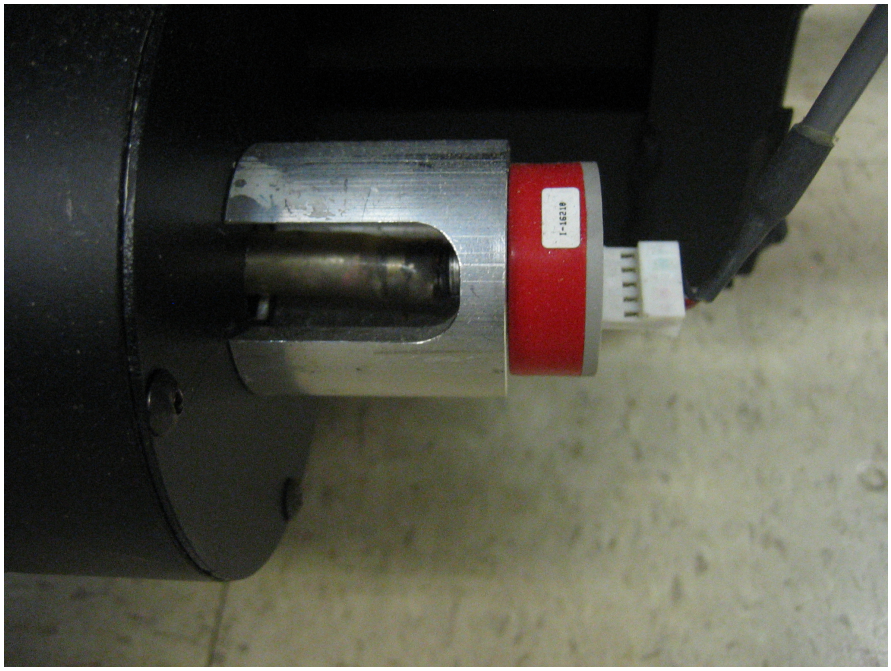

Figure 2.3.3: Encoder mounted and coupled with motor end

The coupling is a simple rod of aluminum machined to taper at one end to fit over the extended motor shaft. The thick end was secured to the motor axle through a pin-hole. The thin end of the coupling could not be directly attached to the encoder because any lateral torsion on the encoder can easily break or damage it. Instead, a length of flexible tubing was super glued to both the encoder and the aluminum coupling. This successfully transferred the rotation of the motor shaft to the encoder, tolerating imperfect alignment. This sensor provided a high-resolution input to the PID feedback loop of the motor controller. Direct sensing on the motor shafts was valuable for speed control of the motors, since these sensors were immune to transmission backlash (and avoided instability problems from backlash in a feedback loop). On the other hand, since the motor sensors did not detect transmission backlash, they were less accurate for computing odometry. The wheel encoders sensed wheel motion directly, and these sensors were superior for computing motion kinematics.

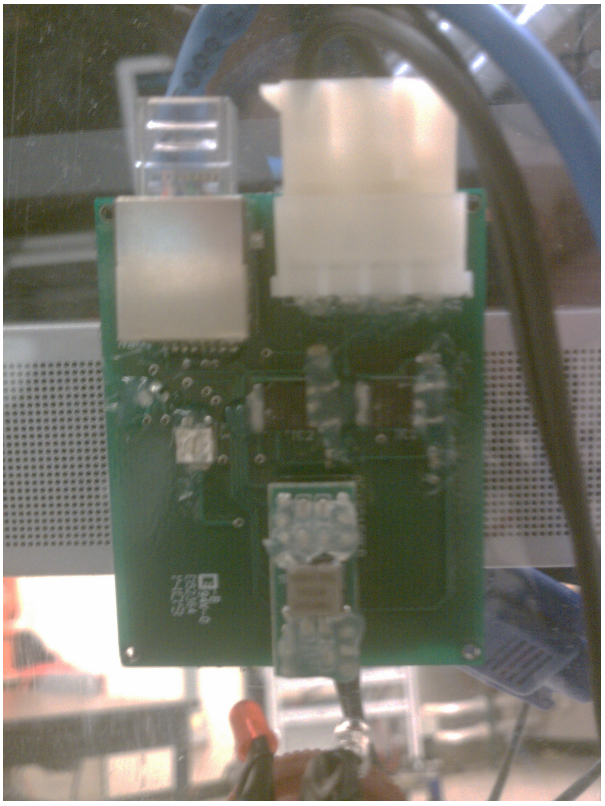

Figure 2.3.4: Analog Devices ADXRS150 Gyro Sensor mounted on the robot

Roberto has a yaw-rate sensor (Analog Devices ADXRS150 Gyro) positioned in the center of its lid. This is a single axis gyro that detects the rate at which the robot turns. It can measure a yaw rate of up to ±2.6 radians per second. It also has a temperature sensor and a reference voltage output. Because wheels slip when turning, an accurate measurement of orientation cannot be determined by wheel encoders alone. The yaw-rate sensor provides additional information that can be used to calculate orientation.

Figure 2.3.5: SICK LIDAR LMS291 mounted on the front of Roberto

The most expensive and important sensor on the robot is the SICK LIDAR unit (LIght Detection And Ranging). The LIDAR can gather range information of objects in a plane outwards from its center. It has a mirror inside of it that spins rapidly. A laser is fired down onto this mirror and it pulses at a set frequency. The infrared light of the laser (905 nm wavelength) reflects off the mirror, hits some surface in the environment, then it reflects back to the LIDAR once again. The LIDAR can detect the light coming back to it and measure how long it took the light to bounce off the object and back again. Using this time and knowing the speed of light the SICK LIDAR can calculate range estimates up to 80 meters with an accuracy of ±1 cm. The device can be configured to change the angular range and resolution. On Roberto these operate at 180 degrees with 181 laser samples per scan (with a

sample at each 1 degree between 0 and 180, inclusive). The module collects 75 scans a second. They are fed into the Intel PC through a serial-to-USB converter (via the LIDAR's serial interface). These scans are invaluable for both collision and obstacle detection and for localization and mapping. Many mapping and localization techniques (discussed in 4.2) can use these scans effectively. The LIDAR does have several shortcomings though. It cannot detect glass, and it has trouble detecting very dark materials (like black jeans). Also, it can give faulty range estimates if its laser pulses hit a reflective surface. Because of this, the LIDAR is not the only obstacle sensor on the device.

Several sonar range finders (Maxbotix Ultrasonic Rangefinder - LV-EZ1) are mounted in key locations around the robot to provide extra sensory data (Figure 2.3.6). Sonar range finders have many benefits over optical based range finders, the most important of which is that they can detect clear, dark, and reflective objects. Because there are objects of all material types in the robot's intended environment these sensors were added. They also have the advantage of being substantially cheaper than LIDAR units and other optical ranging devices. The major drawback of sonar range finders is that they return one range value that indicates there is an object somewhere within a cone pointing out from the range finder. There could be more objects behind the closest object, several objects, or just one small object near the edge of the cone. Because of these inaccuracies, modeling objects detected by the sonar range finders is complicated [2, 41]. There are 5 sonar range finders on the robot: 2 in the front positioned like headlights, 2 on the sides towards the back of

the robot, pointing outwards from the robot, to detect if it can turn safely, and one on the back center, pointing behind the robot.

Figure 2.3.6: A Sonar Range finder mounted on Roberto

Several web-cameras were tested for use with the robot. The web-camera that was chosen was a Logitech Webcam Pro 9000. There has been previous interest in researching computer vision, object recognition, and vision-based localization techniques in the lab. The web-camera will allow future students to use this platform for research in those directions. The web-camera connects to the main computer and is powered by USB. It has a resolution of 1600 by 1200 pixels.

Figure 2.3.7: Thermal camera (left) and Lilliput touch screen (right) on Roberto

The robot is also equipped with a thermal camera (Figure 2.3.7). Thus far this has only been used as a fun "attention getter" for tour groups, but it has the capability of being used for a number of research projects. It is currently interfaced directly to a small (Lilliput) monitor with no other outputs. This monitor has touch input capabilities and could be used for more tasks than it currently is.

2. 3 .2 Components

There are two computers on Roberto. The first is a CompactRIO (cRIO) from National Instruments (figure 2.3.8). This is a 400MHz embedded Power PC computer that runs VxWorks. It has an onboard Xilinx FPGA. The cRIO runs most of the robot's lower level systems. LabView was used to program the FPGA on the device. There are several modules that plug into the cRIO. These modules interface with the FPGA allowing it to interact with other systems and components. The modules Roberto has are the Digital IO module, the Fast Digital IO module, and the Analog Input module. Because changing the layout and compiling an FPGA file is a very time consuming process, the lab has strived to have identical FPGA layouts on all the robots. This creates two minor issues in that the cRIO modules need to be in identical slots across the robots and the modules' I/O pins need to be used for identical purposes or left unused across all the robots. This has not been a limiting factor yet in adding sensors. Thanks to the identical FPGA layout,s the C code that runs on the cRIO Power PC (PPC) is compatible among all robots. This does not mean that the robots all use the same PPC code, but there is a base set of code that is interchangeable among the robots. Having this consistent base system across robots

is beneficial for the lab because any researcher that knows how one robot works on these low levels knows how all of them work, and the systems can be diagnosed easily between robots to detect failures.

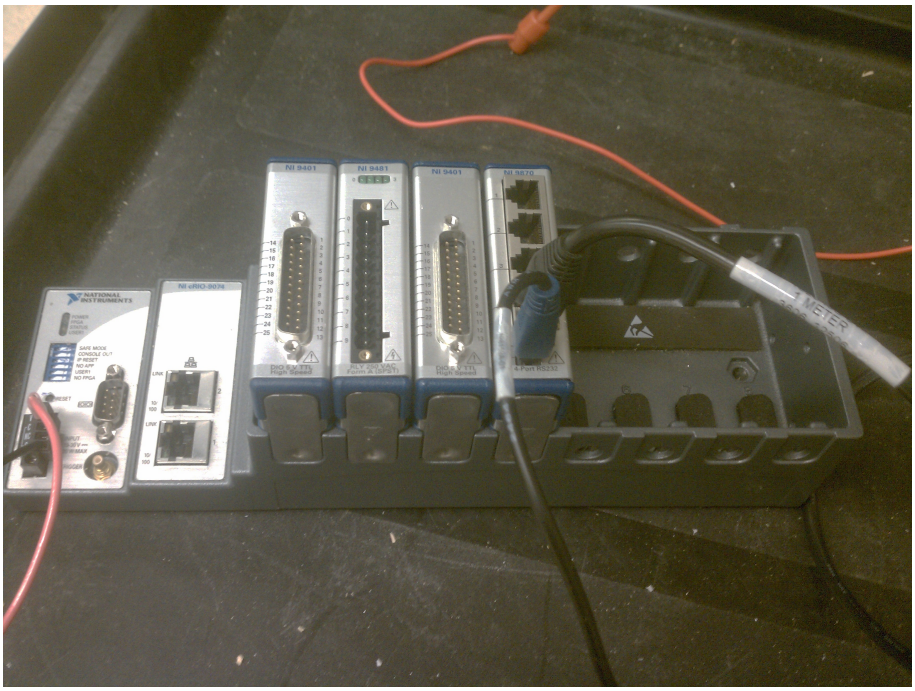

Figure 2.3.8: A cRIO with some expansion slots populated

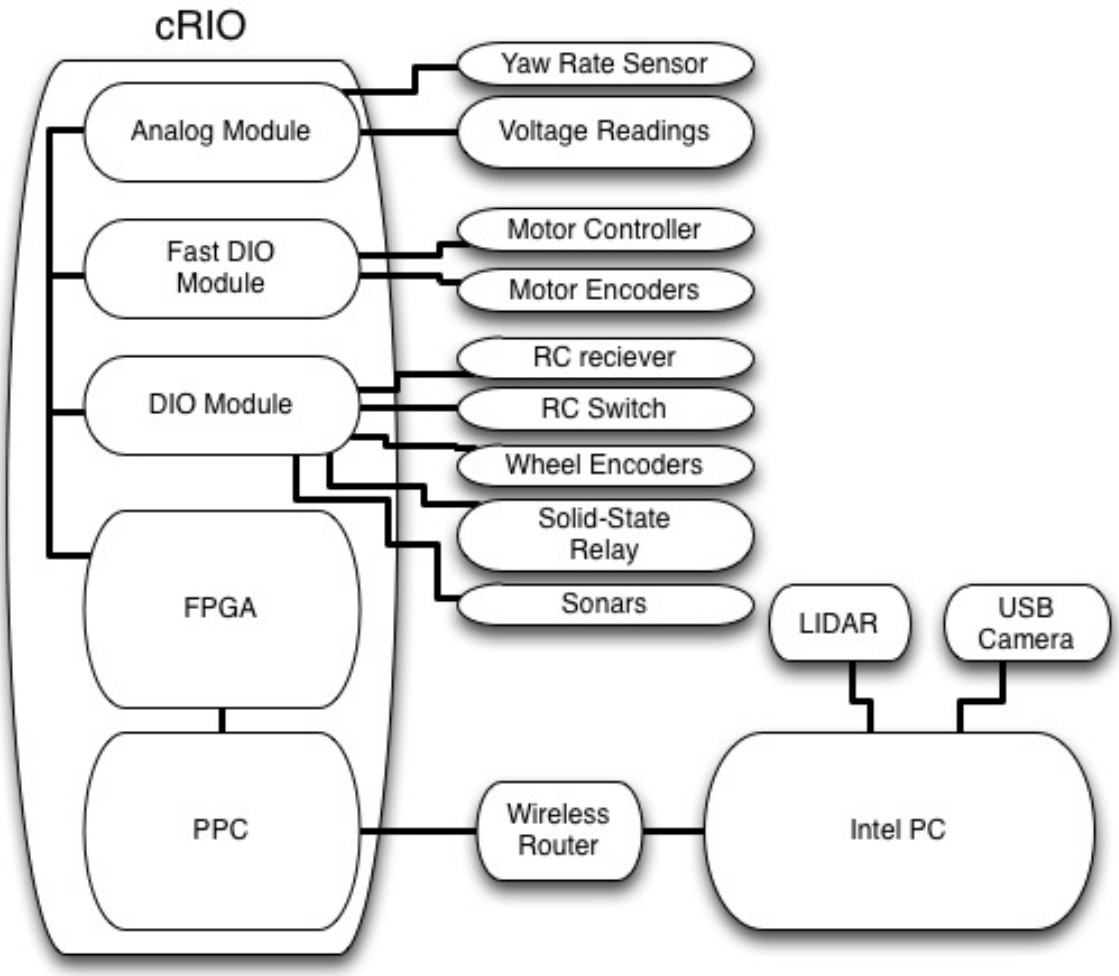

Figure 2.3.9: cRIO and Intel PC sensor interfaces

Figure 2.3.9 shows the cRIO modules and the components and sensors with which they connect and interact. The fast Digital I/O Module is used to read in the motor encoder data and to send control signals out to the Sabertooth 2x25 motor controller. The Digital I/O module is used to: read data from the RC receiver, get the states of various switches and relays around the robot, read wheel encoder data, and send out serial communications. The Analog Input Module is used to read the yaw rate sensor data as well as various system voltages. These include the battery voltage, voltages of the lower 13.8 and 5.0 power rails, the voltage input of the cRIO,

and the voltage of the E-stop logic line. These voltages are divided by 4 before going into the Analog Input Module.

Originally, Roberto had an older model Mac-mini as its main internal computer. Unfortunately, this proved to be underpowered for the computations needed. So a new computer was specified and purchased. The motherboard is a GIGABYTE GA-H55N-USB3, the processer is a 2.93 GHz i7 and 4 Gb of RAM were installed. A low-cost case was acquired to house the computer, but it became apparent all the parts did not fit in the case, so it was decided to leave the lid off and show the computer internals through the side of the robot (Figure 2.3.10). This computer proved to be capable of handling the computations that were required.

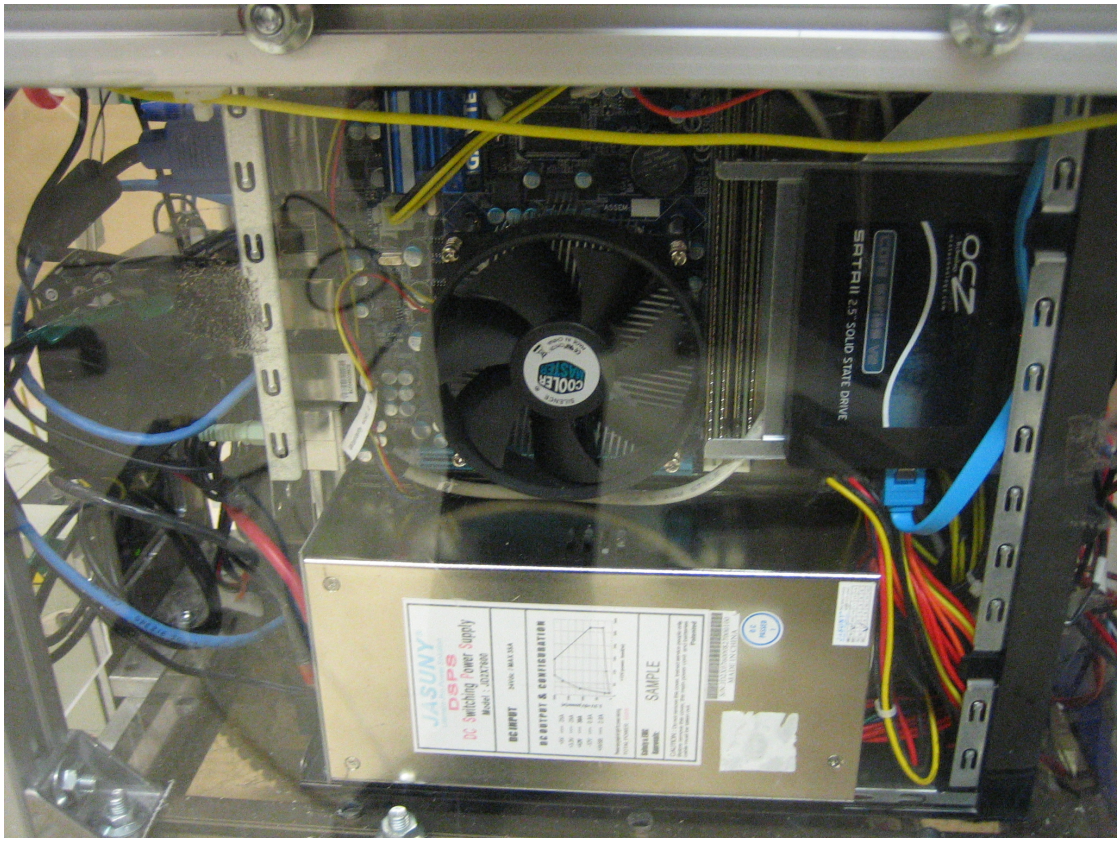

Figure 2.3.10: Roberto's Intel PC with power supply

Each of the robots has a wireless router on board. The cRIO and Intel PC communicate through the network using wired connections. However, it useful for an engineer to be able to wirelessly connect into the network and debug or take control of the cRIO, replacing the Intel computer with whatever computer the researcher is using. There is also interest in having the tour guide interface wirelessly with television monitors to bring up more information for visitors as it goes by (an addition that requires a wireless router or other form of wireless networking).

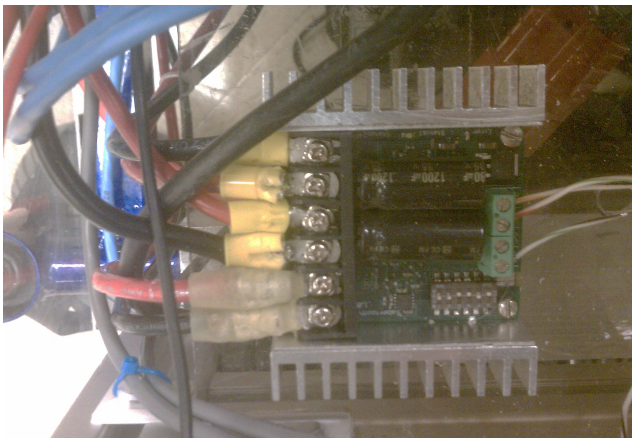

Figure 2.3.11: Sabertooth 2x25 motor controller

This robot uses a Sabertooth 2x25 motor controller [49] (Figure 2.3.11). This motor controller supports four control modes: analog input, R/C input, simplified serial, and packetized serial. Roberto uses the packetized serial mode to control the speeds of each motor. Sabertooth systems have been used on several previous robots in the lab, and they have been found to perform well. It uses an ultra-sonic switching frequency of 32kHz, which means it operates silently without the whine other motor controllers have. It also has thermal and overcurrent protection to prevent itself from overheating or dying from a short circuit. Because of these

features and previous successes, a Sabertooth controller was chosen for use on Roberto.

Figure 2.3.12: Xbox 360 controller (left) and Xtra Speed 2.4 GHz transmitter (right)

Roberto has some components for tele-operation of the robot (Figure 2.3.12). It is equipped with a USB XBox360 wireless receiver that plugs into the Intel PC. This control interface requires some processes to be running on the PC in order to interpret the controller signals and translate them into motor controller commands. There is a further fallback controller, that is an Xtra Speed 2.4 GHz radio control system. A hard switch inside the robot can be flipped to cause the cRIO to ignore all commands from anything except this RC controller. This is a failsafe in the event the robot is in the field and there is a hardware or software failure at the higher (PC) level. This has not been an issue since Roberto's redesign, and in practice the Xbox controller is preferred for controlling the robot manually.

A powered USB hub is connected to the Intel PC. This hub is used for all extra or low-priority systems. These include the Xbox receiver, mice, keyboards, and

other peripherals. The LIDAR and web-camera are connected directly to the Intel PC and not the USB hub.

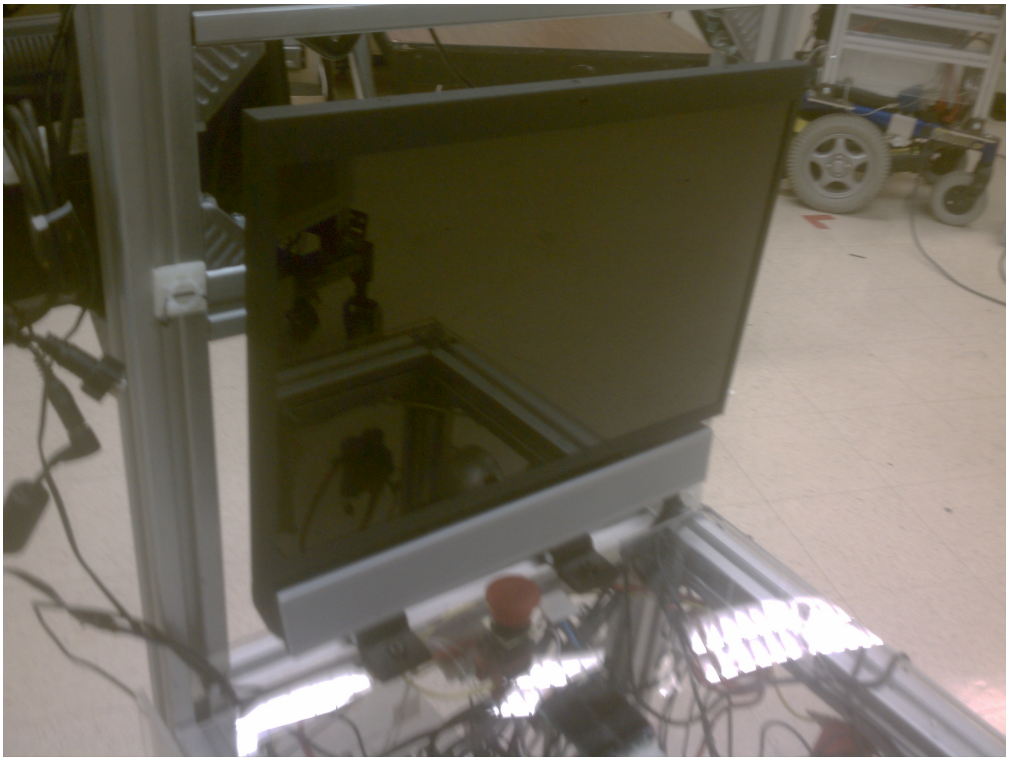

Figure 2.3.13: Flat screen monitor mounted on Roberto

There is one final large flat-screen monitor on the robot that is plugged into the Intel PC (Figure 2.3.13). It is used for both debugging while developing on the robot and for displaying interesting mapping and planning information while the robot is giving tours. Below it are speakers that are powered by the 13.8 V supply. They are plugged into the audio out port from the Intel PC.

Chapter 3: Software Architecture

Roberto's software is divided between 2 systems. One runs on the cRIO and the other is on the Intel PC. The cRIO interfaces with most of the hardware, and the Intel PC executes the higher-level planning and mapping. Localization happens on both systems. The Intel PC uses a framework called ROS (Robot Operating System) [49]. In section 3.1 the cRIO system is covered in detail. Section 3.2 details the requirements of the PC system and options considered for it. Section 3.3 details the ROS architecture used on the PC.

3. 1 Low-Level, Real-Time Control and I/O

The cRIO is comprised of an FPGA, a Power PC processor, and several removable modules. The FPGA interfaces these modules, which in turn interface with most of the hardware on Roberto (Figure 2.3.9).

The Power PC is connected internally with the FPGA and can access specific registers that the FPGA can use to share data with the PC. While LabView was used to program the FPGA, the embedded computer runs a real-time C program that is made with the NI GCC tool chain. National Instruments no longer supports this method and would prefer that LabView be used to program the entire device, but the lab decided against this, because the researchers preferred to work C rather than LabView. The PPC program accesses the yaw-rate sensor, motor encoders, wheel encoders, sonar range finder data, robot voltage values, RC switch state, and RC receiver commands from the FPGA.

The program uses some of the sensor data to create an estimate of where the robot is relative to when the system was turned on using an EKF (Extended Kalman Filter) [42]. An EKF is a signal filter that can accurately take noisy correlated data and create an accurate state estimate. It will be covered in more detail in section 4.1. The cRIO then broadcasts the position of the robot according to the EKF, along with the sonar range finder sensor data, onto the robot's network (via an Ethernet cord connected to the wireless router). The PPC program can be signaled to send out information on the robot's status including voltage values of the various power systems on the robot and the E-stop status.

In normal operations mode, the cRIO expects to get command packets from the network. These packets, which contain a desired linear and angular velocity pair, can come from any device on the robot's network. These velocities are converted in the PPC program into appropriate differential speeds for each motor, and then those speeds are converted to units of motor-encoder tics per delta time (ticks/dt). The ticks/dt are fed into the FPGA from the PPC program where they each command respective PID wheel-velocity control loops The final output of each PID is the motor controller command for its respective motor, which is sent through the Fast Digital I/O module to the motor controller. There is a watchdog timeout on the cRIO such that if it does not receive a command within 100ms of the previous command, then it commands the motors to 0 current, bringing the robot to a halt.. This is in case any controlling device crashes or is unable to send command packets at a fast enough rate to safely control the robot. When the RC override switch is flipped inside the robot, the PPC program changes state to ignore all incoming

commands from the network and forwards direct control from the RC controller to the motors.

3. 2 Higher-Level Computing

The Intel PC receives position-estimate and sonar data from the cRIO and also collects camera images and LIDAR data. In normal operation it is the PC's job to determine exactly where the robot is in the environment, map all obstacles in the environment, determine where the robot needs to go, make a plan to safely get there, execute that plan while handling any possible environmental changes, and interact with humans in the environment in a safe, entertaining and friendly way. There is an alternate operational mode where a wireless USB Xbox 360 controller receiver controls the robot directly.

Previous lab robots have had many different implementations to their higher-level software. This software is really the brain of the robot. Jinx and Harlie had two radically different higher-level software architectures on the PC side when they competed in the IGVC competition in 2010.

Jinx had a large system written mostly from scratch. Using the OpenCV (Open Computer Vision) C/C++ library to make a probabilistic map of the world based on LIDAR data, the system implemented a hybrid bug-and-A* planning algorithm[12, 31] to get from one goal state to the next. A higher level of the software kept track of what goals had been reached, and when a goal was reached it assigned the next goal on a list to the system.

Harlie's system used the ROS framework (Robot Operating System) on a standard Ubuntu Linux operating system. This is an open-source modularized system that allows for ease of parallelization. It is well supported and stable. The system involves creating ROS nodes as independent processes that use message passing to share data among them. There are many open-source nodes available that implement useful steering, mapping and planning algorithms for mobile robotics. Harlie used a combination of open-source nodes and the lab's own planning and steering nodes.

Beyond the higher-level software on previous devices in the lab, many other robot middleware systems exist beyond ROS that could be used on Roberto. Miro, Orca and OpenRDK are all open-source middleware solutions whose communities are defunct [43, 47, 48]. OROCOS has an active community, but is missing many of the feature of ROS. ROS has better documentation and example code than OROCOS [49]. Also, ROS is compatible with Gazebo, a powerful simulator. Due to the lab already having experience with ROS on Harlie, ultimately, the decision of what software architecture to use came down to ROS or the system on Jinx.

ROS and Jinx's system both had their own strengths and weaknesses. Jinx's system benefited from being completely written by the lab, and therefore could be quickly customized to the problems presented at IGVC. Its mapping system was excellent for both static and dynamic environments. However, its planning system was not suitable for dynamic environments. Also, the system was unstable and crashed due to computation errors several times on the course during competition. Though Harlie did not perform as well as Jinx at the competition, this was not

because of the software used on the robot. There were hardware issues, specifically with the motor controller (which was underpowered for the weight of the robot). ROS proved to be an excellent framework for development of mobile robots. The ROS framework gave free parallelization and enforced modularity in program design. It also proved to be stable (the system never crashed). What's more, the simulator Gazebo allows all software to be tested without the robot present. This powerful simulator is especially useful for testing edge cases and robustness of steering and planning algorithms.

There were drawbacks to ROS as well. The core nodes were written for robustness and as such are slower and more computationally intensive than a system that is streamlined for a specific problem. Also, a non-trivial amount of development time needed to be spent learning how nodes work and creating the correct configuration files or altering the nodes to work with the robot.

Given that most of Jinx's system was unable to be used for indoor dynamic environments and that the mapping system was unstable, it was decided that rather than building a completely new system from scratch, ROS should be used as a base framework for Roberto's higher-level software system. As part of this, it was decided to build an architecture from Harlie's existing system that would allow for a shared code base using robot-specific configuration files to run on all robots whenever possible. Therefore Roberto's Intel PC system, like Harlie's, has an Ubuntu Linux OS and is running a ROS framework.

3. 3 Higher-Level Software Environment: ROS

ROS nodes can be created by launching them through the command line, or more commonly by executing the "roslaunch" command with a ROS launch file as input. These launch files can create nodes with specific configurations and execute other launch files. Roberto has a script that is run when the system is turned on that runs 5 ROS commands with sleep commands in between to mitigate race conditions. These commands and what they launch are shown in detail in Figure 3.3.1 (below). Squares in this figure represent ROS launch files; rounded rectangles represent ROS nodes, except in two cases where they are labeled as ROS groups.

The startup script initially executes a roslaunch command, targeting the file "cwru_bringup_no_tele.launch". This launch has been designed to initialize all robots in the lab. It and all subsequent files called within it are the only launch files Roberto has in common with the other robots in the lab. It will not start any AI, planning or mapping nodes, but it will set up all lower-level configurations. The "no_tele" suffix means "no tele-operations". There is another version of this file called " cwru_bringup.launch" that will start nodes to allow the robot to be controlled by an XBOX 360 wireless controller, allowing for tele-operation of the robot. This file simply launches another ROS launch file named "start_base.launch". Other experimental launch files can be started in this file, as is noted on figure 3.3.1. However, Roberto does not launch any experimental files here.

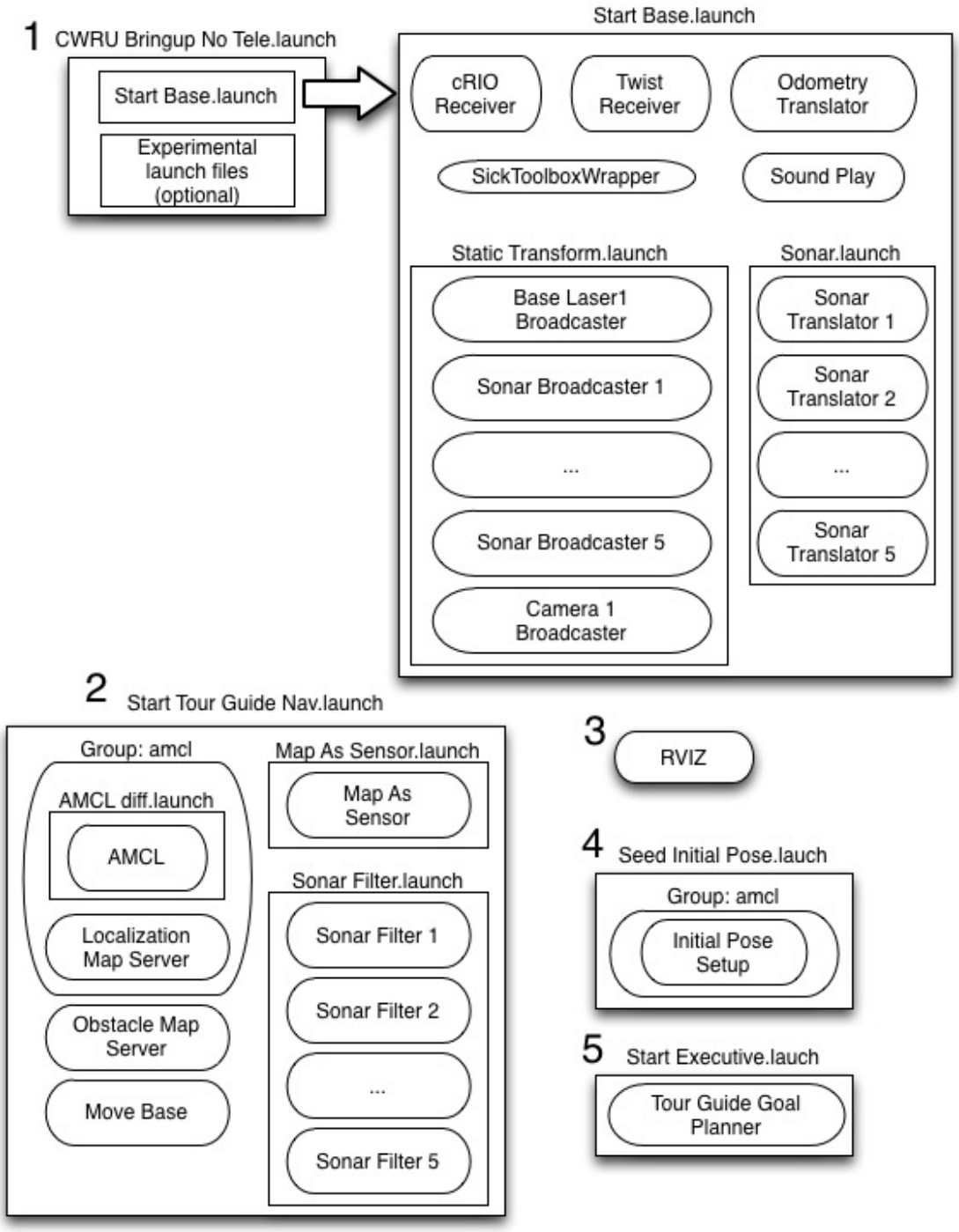

Figure 3.3.1: roberto_launch.sh startup script shown as a chart

The "start_base.launch" file creates five ROS nodes and launches two more launch files. The first node is the cRIO Receiver node. This node listens on the network for robot "pose" (state) of the robot (position, velocity, variance, etc.). It also strips the sonar values from each received packet. The pose and sonar values are broadcast to other ROS nodes

The ROS library has a package called TF, which stands for “transformation.” This package allows the user to create a tree of transformations to keep reference frames bound to each other. This way if the transformation from the center of the robot to a sensor is known, the TF package can instantly convert the data the sensor gives from its coordinate frame to the coordinate frame of the robot, or to any other coordinate frame in the TF tree.

The second node started by "start_base.launch" is the Odometry Translator. It listens for pose broadcasts from other ROS nodes and converts the cRIO pose data into the ROS network's robot-centered coordinate frame, known as "base_link". The "base_link" frame's origin is the center of the robot. This transformed data is then broadcast as odometry information. Then the Odometry Translator updates the TF tree with the translation of the cRIO's EKF output with respect to the "base_link" frame.

The ROS framework refers to movement as “Twists.” Anything involving velocity is stored in Twist-related data structures. When the system wishes to command velocity to the wheels it does so with Twist commands. The Twist Receiver node listens for these commands and forwards them along to the cRIO via the network. This is the third node launched by "start_base.launch".

The Sick Toolbox Wrapper node is a powerful node that encapsulates the USB-serial drivers of the SICK LIDAR. It handles all interactions with the hardware and broadcasts sensor data to the ROS network when it is received. The Sound play node simply allows ROS to play sounds on the device's hardware. These nodes are also launched by "start_base.launch".

Which "static_transform.launch" and "sonar.launch" files that "start_base.launch" launches are robot dependent. On each robot the ROBOT environment variable is set to the robot's name (Roberto, Harlie, etc). The "start_base.launch" file looks in the "cwru_configs" folder and launches the files that are in the folder with the same name as the ROBOT environment variable. Only Roberto's files will be covered here.

The purpose of the "static_transform.launch" file is to propagate the TF tree with proper transforms for each sensor on the robot. Roberto's "static_transform.launch" file launches a broadcaster node for each sonar (five). These nodes each broadcast a conversion from its sonar's coordinate frame to the "base_link" coordinate frame. The file also launches a broadcaster to convert from the LIDAR's coordinate frame to "base_link", and a broadcaster for the USB camera's frame to "base link".

The "sonar.launch" file launches "sonar_translator" nodes. These nodes listen for the sonar broadcasts from the cRIO Receiver node and convert these range values into a laser-scan data. This data is generated as if a LIDAR had scanned the field of the sonar, and found range values at all points in the scan equal to the

distance reported by the range finder. Thus far this is the best method the lab has created for handling sonar data. In the future this method is likely to change.

The next launch file Roberto's startup script launches is "start_tour_guide_nav.launch". This file launches the navigation section of Roberto's ROS network. All mapping, planning, localization, and interaction nodes are launched from this file.

Roberto uses the AMCL (Augmented Monte Carlo Localization)[42] algorithm for localization. This will be covered more in detail in section 4.2. In short, it is a localization method that uses a known map of the environment, a movement model of the robot, and sensor input, and uses a particle filter to compute a best guess for the pose of the robot based on these inputs. There were many other localization options that could have been pursued, such as SLAM (Simultaneous Localization And Mapping)[42] or simpler landmark-based localization methods (discussed in section 4.2). However, given that Roberto was going to be in a known environment and that AMCL is an extremely robust algorithm, it was deemed to be the best choice.

ROS can separate nodes into namespaces. These namespaces are called "Groups". Roberto's system uses only one group beyond the default. This is the AMCL group. The "start_tour_guide_nav.launch" launches both "amcl_diff.launch" and the localization_map_server ROS node within the AMCL ROS group. The file "amcl_diff.launch" launches an AMCL ROS node with the proper configurations for the SICK LIDAR on the ROBOT. This AMCL node only uses the sensor input from the LIDAR. The sonar data could theoretically contribute to AMCL. However, the current

node has not been written with this functionality. The lab has considered improving upon this in future work. Though due to the coarse data generated by each sonar and the small number of sonar range finders on the robot, it is unlikely that they will contribute substantially to the AMCL algorithm.

The localization_map_server, launched by "start_tour_guide_nav.launch", creates a ROS map server that holds an *a priori* map of the environment. These localization maps are recorded using the Gmapping ROS package. Gmapping uses a laser-scan-based SLAM algorithm to map an environment. It outputs maps as large image files that the map servers then access for localization.

A second map server node is launched in "start_tour_guide_nav.launch". It is the "obstacle_map_server" ROS node. This server uses a map that contains all *a priori* obstacles in the environment, which is a different map than the localization map. The "start_map_as_sensor.lauch" file launches a "map_as_sensor" ROS node. This node will republish the "obstacle_map_server" node's map as a point cloud sensor. The details of why this is done are covered in section 4.1. However, the reason the AMCL map server and nodes are confined to a separate group is because there are multiple map servers.

The "move_base" node is the main planning node provided by the ROS package. This node is created and interacts with the obstacle_map along with all the sensor data. It maintains 2 cost-maps and has 2 planners: a global planner and a local planner. When the node is given a goal, the global planner picks a general route to get to the goal, while the local planner deals with obstacles and steering to try to roughly accomplish the global plan. This planning is covered in more detail in

section 5.2. This node takes in many configuration files and values to customize it for Roberto's frame and movement capabilities.

Lastly, "start_tour_guide_nav.launch" launches the file "sonar_filter.launch". This file launches a "scan_to_scan_filter_chain" node for each sonar range finder. These take the sonar data as laser scan data and propagate the data to cost-maps. It should be noted that at the time of this writing the sonar data is not propagated to the final planning cost-maps. There are unsolved issues with how to handle the sonar data in a stable way.

The startup script next launches RVIS. RVIS is a ROS visualization tool. It shows map, cost-map, and planning data published from the ROS nodes. This is displayed on the large monitor while the robot gives tours. It also is extremely useful for debugging and testing.

After all of the previous nodes have been set up, the startup script initializes the robot's position. It does this by launching the "seed_initial_pose.launch" file. This file launches an "initial_pose_setup" node within the AMCL group. This node initializes the particle filter used by AMCL to a Gaussian cloud around the starting position of the tour. This allows the robot to very quickly localize as it starts its tour because it does not have to consider the possibility of being started very far away from the default start location.

The final command of the startup script launches the "start_executive.launch" file. This file launches a "tour_guide_goal_planner" node. This node determines what positions the robot should go to and how long to wait at those positions. It also issues the speech commands to the robot. Currently, speech is handled by playing

prerecorded files using the VLC [51] library bindings. The files were created with the Festival [45] text to speech software. This modular system allows for a more complex system to be implemented alongside it. This is covered in section 6.1. All human interaction is handled in this node.

The architecture of the robot succeeds by being safe, stable, robust, portable, modular, easy to modify, and easy to interface with. It has been shown to be portable by running on many different robots in the lab. This design can be used for many applications. Sensor integration has been streamlined through the use of either connecting over USB or using the cRIO FPGA. Its modularity makes it easy to modify or swap out specific subsystems. This is in part thanks to using ROS nodes, and also thanks to the overall hardware and software design.

Chapter 4: Localization

Localization is a well-known problem in the field of mobile robotics. Accurate localization in an environment is crucial for path planning and goal finding. When choosing a localization system for Roberto many established approaches were analyzed. Specifically, using an EKF [42], using LIDAR based SLAM [42], using LIDAR based AMCL [42], and Norman Prokop's camera based localization models[28]. In the end it was found that the best approach was to use an EKF to provide a local pose estimate for the robot, and to feed that into a LIDAR-based AMCL algorithm. This proved quite effective.

4.1 Extended Kalman Filter

The EKF position can accumulate drift over time because none of the sensors feeding into the EKF give a global position (like GPS). The other lab robots are equipped with GPS sensors, which clear the drift from their EKF output (when operated outside).

Roberto's EKF system was leveraged from the existing system on Harlie. It runs on the cRIO PPC. This system provides a pose estimate relative to where the robot was turned on. It is susceptible to small amounts of accumulated drift over time. Jinx and Harlie at times were equipped with GPS and compass sensors that fed into their respective EKF systems. This allowed them to use this system as their only localization in outdoor environments. However, because Roberto operates indoors

where these sensors are useless, it cannot depend on GPS and compass signals for localization.

The EKF system reads in values from the wheel encoders and the yaw rate sensor. It outputs a pose value for the robot's center. Figure 4.1.1 shows the algorithm used in the EKF.

Input: x_{t-1}, P_{t-1}, z_t
Output: x_t, P_t

1 $\hat{x}_t = f(x_{t-1})$
2 $\hat{P}_t = G \cdot P_{t-1} \cdot G^T + Q$
3 $\hat{P}_t = \texttt{ZeroOutBiasXYThetaCovariance}(\hat{P}_t)$
4 **foreach** $z_t^i \in z_t$ **do**
5 $\quad y_t^i = z_t^i - h(\hat{x}_t)$
6 $\quad S = H \cdot \hat{P}_t \cdot H^T + R$
7 $\quad K = \hat{P}_t \cdot H^T \cdot S^{-1}$
8 $\quad \hat{x}_t = \hat{x}_t + K \cdot y_t^i$
9 $\quad \hat{P}_t = (I - K \cdot H) \cdot \hat{P}_t$
10 **end**
11 $x_t = \hat{x}_t$
12 $P_t = \hat{P}_t$

Figure 4.1.1: Roberto's EKF algorithm. Leveraged from Harlie [27].

Here the previous state estimate is x_{t-1} , the covariance of the previous state is P_{t-1} and the new measurements for this iteration are z. The outputs are the current state x_t and the current covariance P_t. In line 3 of the algorithm, the prediction covariance between the yaw-rate bias b_ω and x, y and Θ is cleared. Without this step the x, y or Θ terms would cause the bias to quickly become unstable. This system is covered in detail in [27].

4.2 SLAM and AMCL

SLAM (Simultaneous Localization And Mapping) is a powerful algorithm commonly used by many mobile robots. There are many pre-built packages of this algorithm that can be leveraged. In the ROS framework there is a LIDAR SLAM package called GMapping. This package is a wrapper for the GMapping openSLAM algorithm [9]. In short, this algorithm takes in LIDAR data along with pose estimates, then, using particle filters, it creates a map of the environment and localizes the robot within that map. The maps this method generates can be saved as image files and reloaded later.

This SLAM method works best in primarily static environments. Some moving obstacles are acceptable, but the mapping does not mark obstacles as static and non-static. Though dynamic obstacles do not meaningfully influence the algorithm's ability to localize, the maps generated can have artifacts or missing information due to a dynamic obstacle occluding static features. This algorithm was tested on the 3rd floor of the Glennan Building at CWRU with no dynamic obstacles present. The output map from that test is shown in figure 4.2.1. Since there were many glass walls and doors present, the map acquired falsely declares some walls as navigable space. This had to be corrected manually.

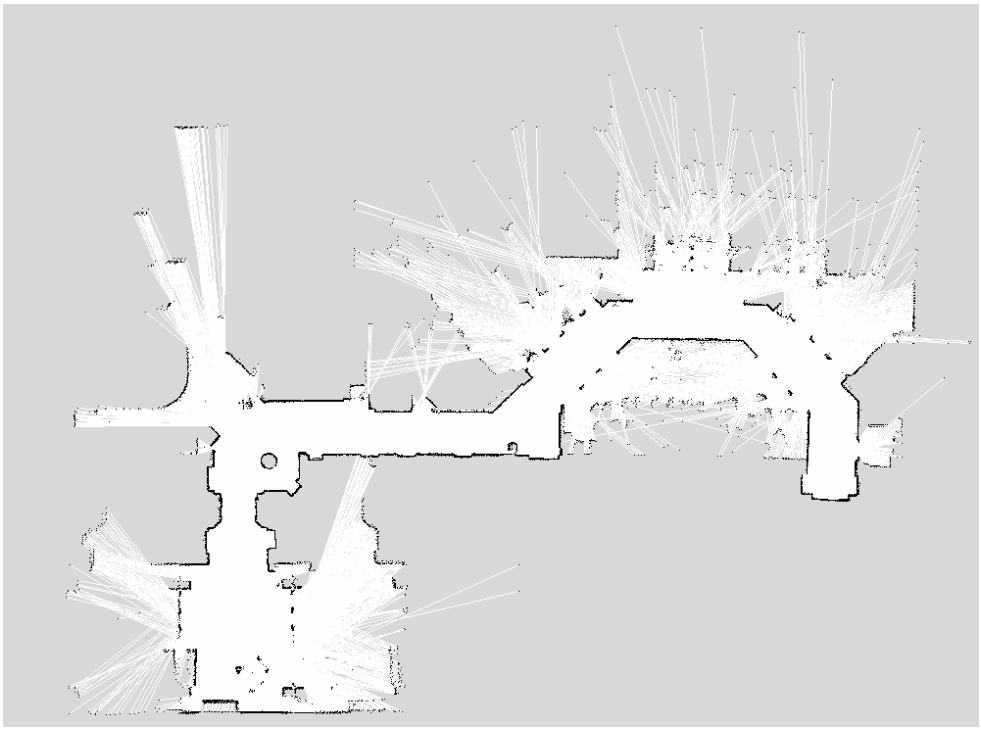

Figure 4.2.1: Map generate from GMapping package of Glennan 3rd floor

The AMCL algorithm is a powerful localization algorithm when dealing with a known environment. ROS provides an AMCL package that can use the maps generated by the GMapping package as reference maps for AMCL.

The AMCL algorithm in ROS uses a particle filter [42] on LIDAR scan data. It creates multiple hypotheses (particles) and compares the error between a virtual scan at each hypothesis compared to the measured scan. The most likely hypothesis is accepted for that time step. The particles are simulated forward by the robot's movement between filter iterations. In the case of Roberto, this change in pose is estimated by the Kalman filter on the cRIO. The particles are resampled, with replacement weighed on likelihood, to form a new particle cloud. There is a feedback in the resampling; if there is a large error across all particles and the

measured results, then random pose hypotheses (particles) will be generated in the state space.

The possibility of feeding the AMCL position output into the Kalman filter was considered. There was difficulty in synchronizing clocks between the cRIO and the Intel PC, along with feeding old position data in to the Kalman filter. This was not pursued further because no system on the robot uses the Kalman filter as an absolute location. The data is only used as pose estimates relative to the previous estimates. Harlie uses a similar system.

The maps generated by GMapping can be used directly by AMCL. However, there are many dynamic objects in the environment that were present in the resulting maps. Non-static objects included chairs and desks in the classrooms that the LIDAR could sense through large glass walls. To minimize unnecessary error or uncertainty in the AMCL module, the GMapping output maps were manually edited to only show static environment features (Figure 4.2.2). Expected free space (white) and walls (black) are used by AMCL for localization, but the grayed areas, representing unknown regions, do not contribute to computing localization. By graying out movable objects (e.g. furniture), AMCL does not attempt to use these as landmarks.

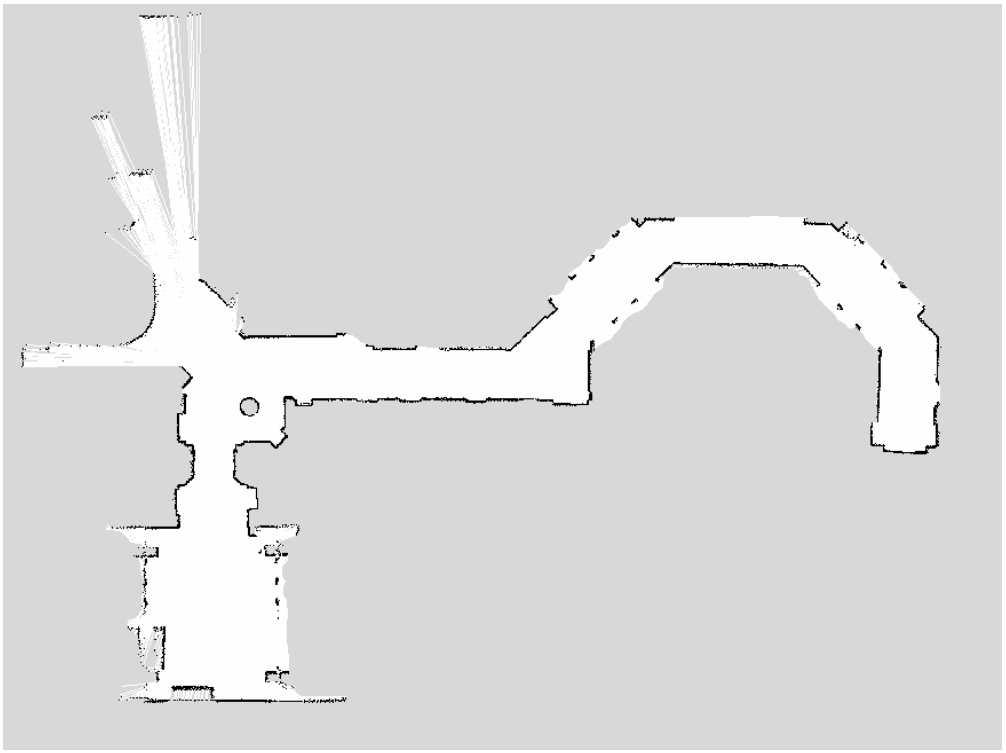

Figure 4.2.2: Map edited to only show static environment features.

There were concerns that because the AMCL module uses only LIDAR data for localization that it would be a poor choice for localizing while giving tours to prospective students. The worry was that people actively walking through the hallway would occlude the features that the localizer expected to see and this could cause AMCL to perform poorly. In practice this proved not to be the case. AMCL performed admirably in test situations. It is a robust algorithm that can handle noisy and error prone data.

4.3 Vision-based Mapping and Localization

Visual mapping and localization was another possible method considered for Roberto. Visual SLAM is a viable approach for indoor mobile robots [5, 16 ,26, 36].

At the time this project was underway ROS had no stable visual SLAM module. There were also concerns that Roberto's on-board computer could not handle the overhead of running visual SLAM and AMCL simultaneously. Monocular SLAM and Stereo SLAM, using SIFT (size invariant feature detection), are also popular visual mapping and localization techniques for indoor mobile-robotics [5, 16 ,26, 36]. However, the camera on Roberto is intentionally upward facing to minimize the noise that people and other dynamic objects would add to the images it collects. Monocular and Stereo SLAM using SIFT work best with forward facing cameras.

Several simpler visual mapping and localization methods were considered instead of visual SLAM. Norman Prokop (a previous graduate researcher in the lab) simulated several linear regression based visual navigation algorithms. These algorithms took in fiducial locations in the image along with camera calibration, camera pose data, and odometry data for each image, and could produce a map and localization of the robot within that map. Because these algorithms used linear regression and not particle filters, they considered only one hypothesis and were less robust than visual SLAM. However they require much less computation. Work was started to implement these algorithms but it was ultimately abandoned in favor of LIDAR-based localization.

Before attempting to implement the Prokop linear regression techniques, experimentation was done with simple image stitching and warping to create a map of the ceiling. In this process, first, the camera was calibrated to find its extrinsic and intrinsic parameters. Then the image was projected into the plane of the ceiling. The hope was that if all calibrations were correct then a smooth mapping of the

ceiling would be possible by overlaying camera images transformed by LIDAR AMCL positioning data. Figure 4.3.1 shows the results of this process. The extrinsic camera parameters were not accurate enough to succeed in this. The image shows a mapping of the robot going down the hallway to the end on the right side, then turning around and going back. Many of the lights are cut in half and the other half is several feet off from when the robot was passing it the opposite direction.

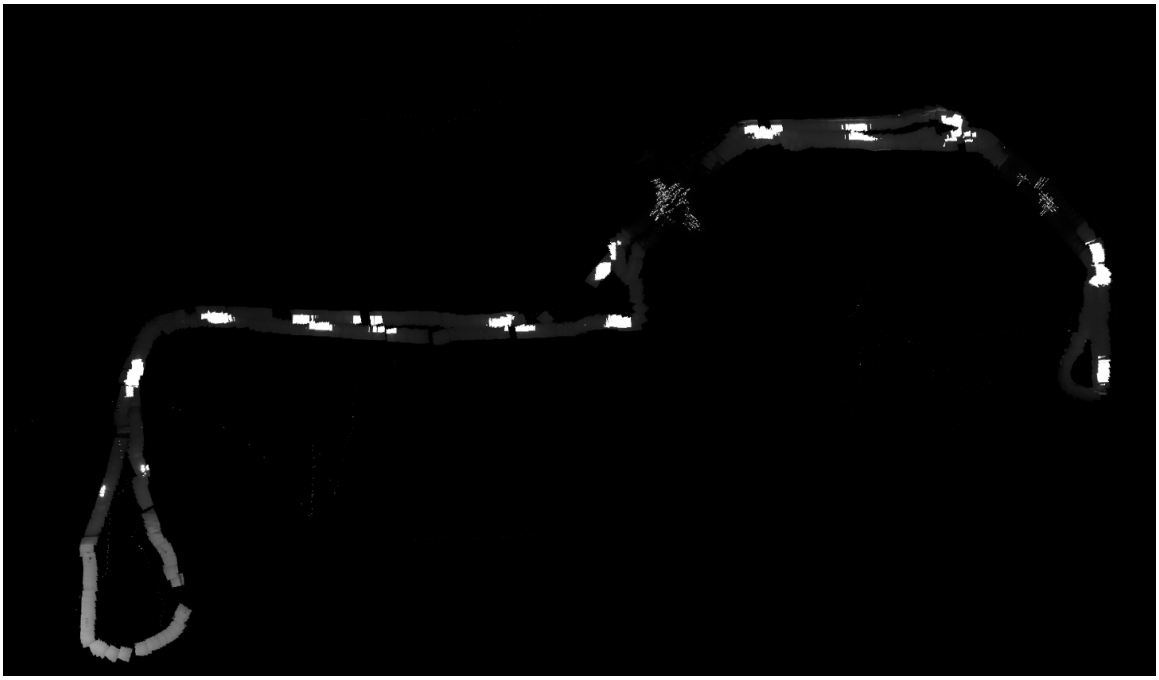

Figure 4.3.1: Visual stitching mapping of the hallway.

Simple image matching was to be used with this image-stitched map to localize the robot on the map and then calculate the robot's position in the hallway from that mapping. This was quickly abandoned as the calibration process could not be perfected enough to make these efforts worthwhile.

In order to test the Prokop method, first feature/fiducial detection was necessary. This was done by thresholding the images so only the lights were visible,

then running a Harris corner detector on the image. Image flow data was used to track features from frame to frame. When a feature left the frame however, it if came back into frame it would be treated as a new feature during the initial mapping process. Features were successfully detected as the corners of the square lights in the hallway. However, reflection in the glass through much of the hallway generated false features, and decorative LED resistor and capacitor lights generated unstable features. There were also detection issues with round lights in the lobby area. It was suggested that there be configurations to ignore camera images at those areas of the hallway. This was not pursued.

Aside from feature detection issues, the camera was calibrated by hand and could easily be bumped by tourists or researchers. An auto calibration procedure was suggested but was never implemented. It would also require auto calibration mid-tour, which would be unsafe. The camera also could not see more than one light at a time, and in traveling down the hallway most of the frames had no features in view. This would require relying heavily on odometry data until a new feature became visible. When a new feature would come into view, the position would snap suddenly from the odometry estimate to the visual localization estimate. This could cause unstable steering commands and unsafe movement from the robot.

To complete this visual localization work, new ROS nodes were created. ROSBAG recordings, a data-recording file for ROS, were made to capture vision and localization data [49]. This proved to be useful for fine-tuning feature detection. A ROSBAG file could be replayed multiple times to replay the data of the robot and new ROS nodes could listen to that data and analyze it in real-time. Without this

functionality the robot would need to be manually driven through the environment to test every change to various vision algorithms.

Ultimately visual localization was abandoned before more progress could be made on it. There were many drawbacks specific to the environment, camera, and robot used that made it a poor primary localization method for this platform. Visual localization is still a powerful and useful tool in robotics, but, for Roberto, AMCL localization with LIDAR data was found to be a superior method of localization for tour-guide operations. However, Roberto proved to be a convenient platform with which to research vision localization.

Chapter 5: Planning and Motion

For a robot to navigate a dynamic environment safely, it needs to be able to map its surroundings and plan and execute a path through the environment. Several approaches to these problems were considered for use on Roberto.

5.1 Mapping Obstacles in the Environment

Two obstacle-mapping approaches were considered for use on Roberto. The first was using the ROS costmap_2d implementation. The second was to leverage a mapping system written for Jinx.

The ROS costmap_2d module builds a fixed-resolution occupancy grid from sensor data. It inflates obstacles in the grid to mark off areas the robot cannot safely navigate according to configuration files that specify the dimensions of the robot. It generates a cost-map that marks areas closer to obstacles as higher cost. This can be used by planners to preferentially stay away from objects. Figure 5.1.1 shows this costmap, where black is a detected obstacle, grey is obstacle inflation, and the coloring shows the cost of areas around the obstacle (violet is high cost and red is low cost).

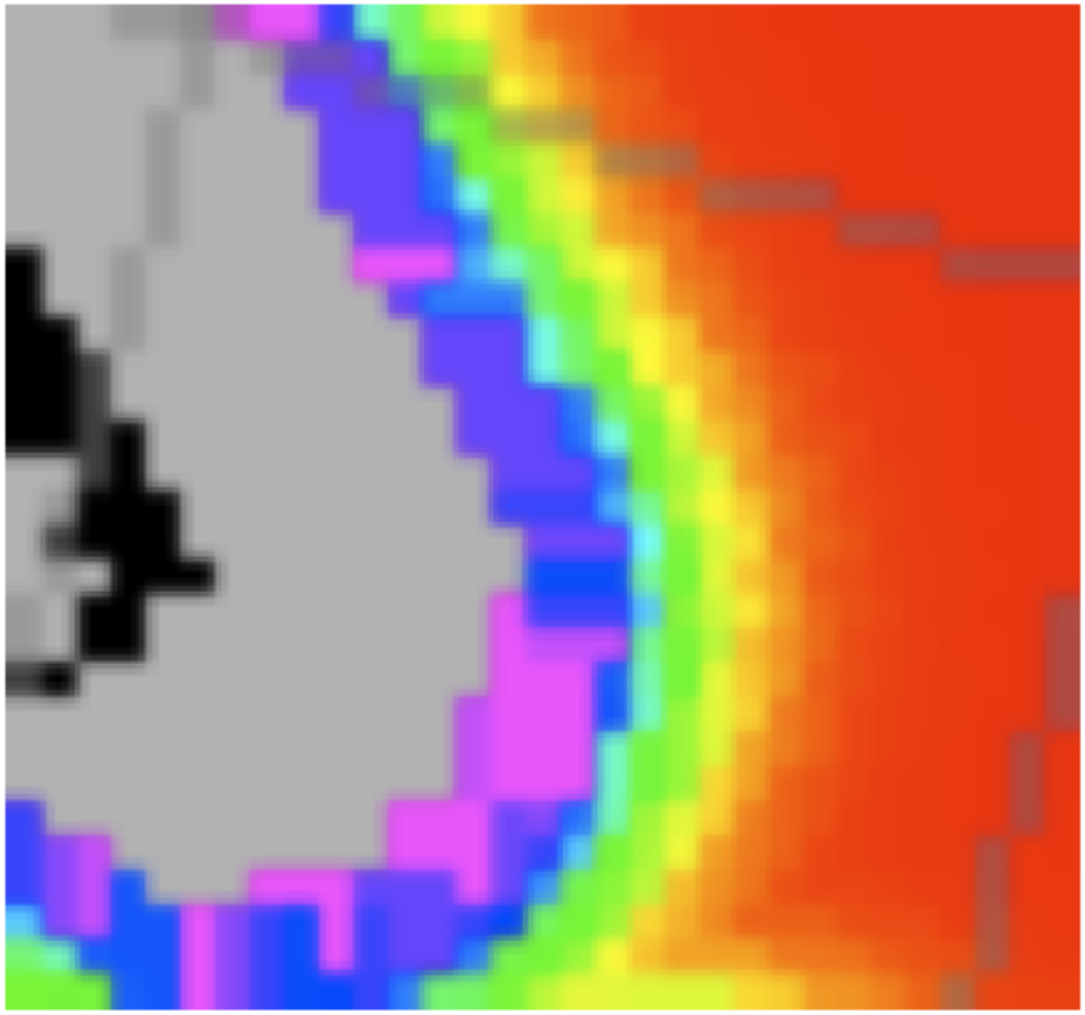

Figure 5.1.1: Section of costmap_2d generated costmap

There are known shortcomings to the default ROS costmap_2d implementation. A major shortcoming this project faced was there was no way to specify *a priori* obstacles to the robot through a map that could not be cleared by sensor data. The ROS module behavior is to remove obstacles when laser scans show the area to be clear. In the Glennan 3rd floor there are several large glass panels and benches that are invisible to the LIDAR on the robot. A way to ensure the robot would not run through them, when it is accurately localized, was needed. The other shortcoming faced was that the ROS occupancy grid implementation never

forgets old data; it must be cleared by LIDAR scans. Because of the discrete nature of the LIDAR scan pings, mapped obstacles lying between sequential pings are not cleared. This causes old obstacles to remain as artifacts that should be cleared, but are not. In practice this can lead to the robot taking awkward paths to get around obstacles that do not exist. Other researchers in the lab experienced issues with this approach when applied to precision navigation. A fixed-resolution cost-map is not ideal because there can be areas that require high resolution for navigation and others that do not. This problem fortunately did not manifest in this project.

The second mapping approach was developed on Jinx. It uses an occupancy grid kept in a bitmap image. It requires an accurate position estimate of the robot in a global frame. The bitmap occupancy grid has a pixel to cm^2 conversion and is initialized to a grey value of 128 (where the bitmap can hold values of 0 to 255 with no overflow). LIDAR scans are read in and rotated to match the heading of the robot relative to the global frame. A polygon is masked onto the image representing the area within the LIDAR scan. A clearing constant (Cl) is subtracted from the occupancy grid within this mask. Then a second mask is constructed by drawing circles corresponding to each LIDAR ping into a temporary buffer. A second constant (I) is added to the occupancy grid within this mask. This process is repeated for each new LIDAR scan. The bitmap becomes a probabilistic map of likely locations of obstacles built up over all previous scans. In practice, a threshold would later be applied to the map to have a binary determination of free space and obstacles for planning algorithms.

Figure 5.1.2: Example masks for clearing (left) and incrementing (right) in the mapping algorithm developed on Jinx

This approach was separately tested on a UAV platform. In this test, each pixel represented a 15 mm by 15 mm area. A 2.53 MHz Intel quad-core i5 with 3 GB of memory was used. The mapping algorithm took 5.6 ms per iteration. Scan data was loaded from a file, removing the delay from waiting on the next LIDAR scan. At 5.6 ms, this method is quite fast for a mapping algorithm [52].

There were many advantages to Jinx's mapping approach. It is computationally simple. It also was trivial to add a global forgetting constant every update that would allow old data to be forgotten after N scans. One would simply subtract a constant from the bitmap every update. This approach is also fast, and would take less CPU overhead [52].

However, there were also drawbacks to the Jinx approach. It did not lend itself for use with other ROS nodes. A completely new planner would need to be written to take in the images that the mapper produced. Also, the Jinx code was hastily written for a competition. It was not well documented, and it had several

bugs that would cause unexpected crashes. This lack of stability and testing in the code was a large safety concern.

Ultimately, it was decided to try to fix the problems with the ROS mapper rather than to try to integrate the Jinx mapper with ROS. The inability to set permanent *a priori* obstacles that cannot be cleared was the most important issue that needed to be fixed with ROS. In many robotics applications it may be necessary to define areas that the robot cannot go where the robot has no way of sensing these areas other than *a priori* knowledge. Some places a robot would need to avoid that are undetectable can include trap doors, pit-traps, pressure sensors, and laser-alarmed areas.

This issue was solved by creating a second map server for known obstacles and rebroadcasting that map as a sensor. The ROS sensor model allows for sensors to map in 3 dimensions, so the map is broadcast at a z height different from the LIDAR and sonar sensors. This change in height prevents Roberto's on-board sensors from clearing obstacles specified by the *a priori* map.

Two approaches were taken to generating the obstacle maps. First, the blueprints for the 3rd floor of Glennan were carefully measured and digitized to create a map (figures 5.1.3 and 5.1.4). For the second approach, the output of the Gmapping SLAM process used for localization was duplicated and edited in an image editor and obstacles were added to it (figure 5.1.5).

Figure 5.1.3: Blueprint of the Glennan Building 3rd floor.

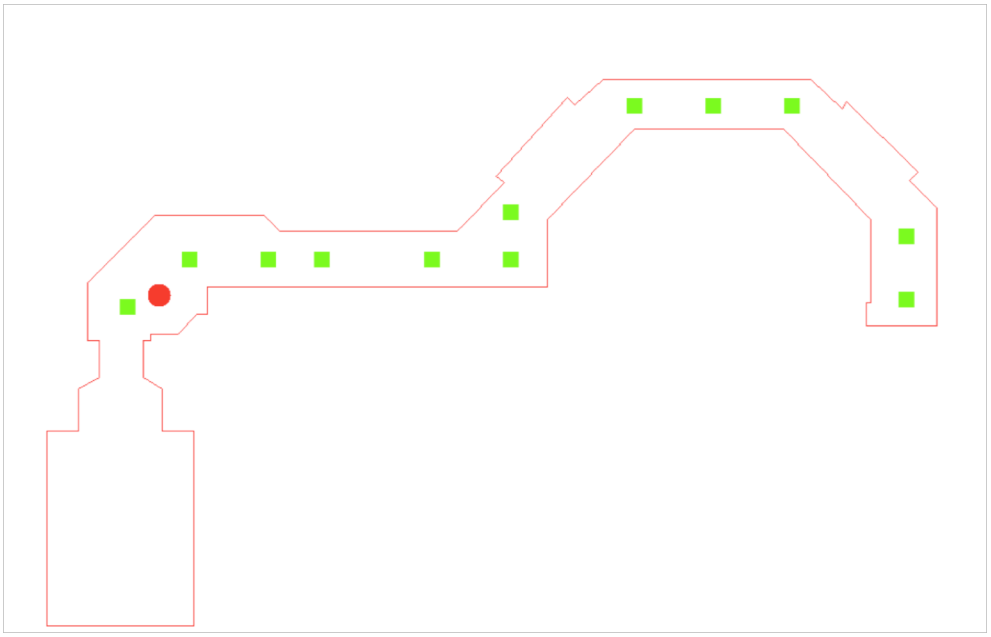

Figure 5.1.4: Digitized blueprint of the hallway through which Roberto travels

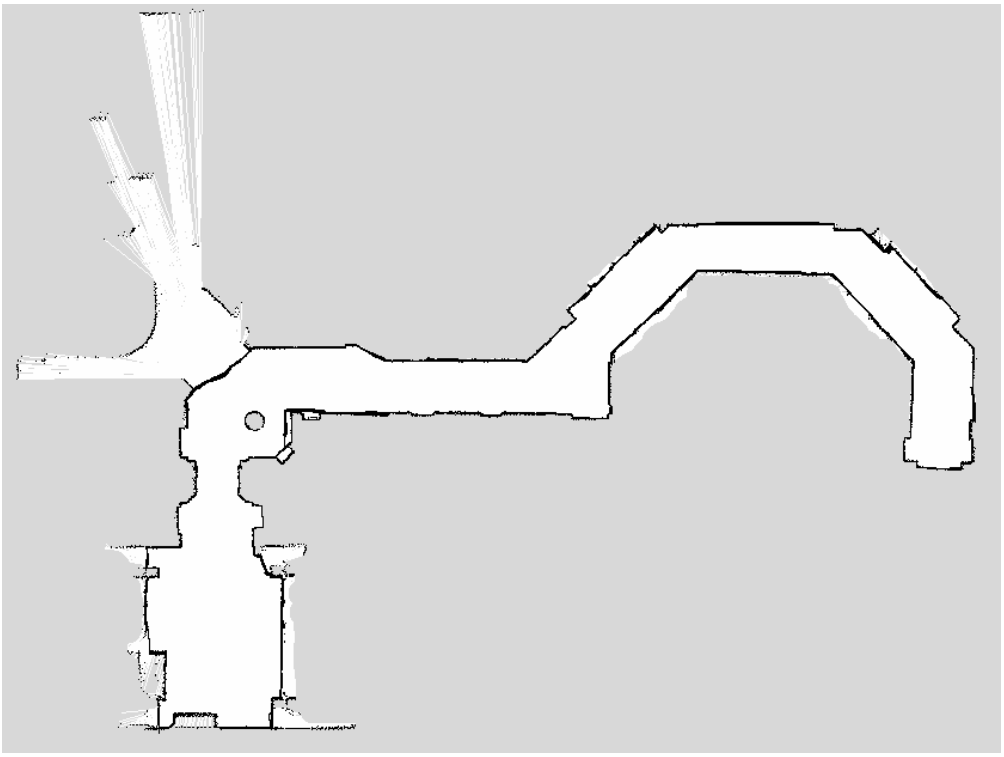

Figure 5.1.5: Obstacle map generated from the Gmapping localization map, edited to include glass walls.

The accuracy of the sensor measurements generated by the obstacle map fake sensor is extremely dependent on the robot having an accurate estimate of where it is in the map. A comparison was done between the digitized blueprint map and the SLAM generated map (figure 5.1.6). The SLAM map and the blueprint differ by almost half a foot in several places. This may be due to drift in the generation of the map with SLAM or due to actual differences in the building construction and the blueprints. Regardless, the AMCL module localizes the robot in the map generated by SLAM. Therefore, the greatest accuracy for static *a priori* obstacles was achieved by using the modified SLAM output map and not from using the digitized blueprints.

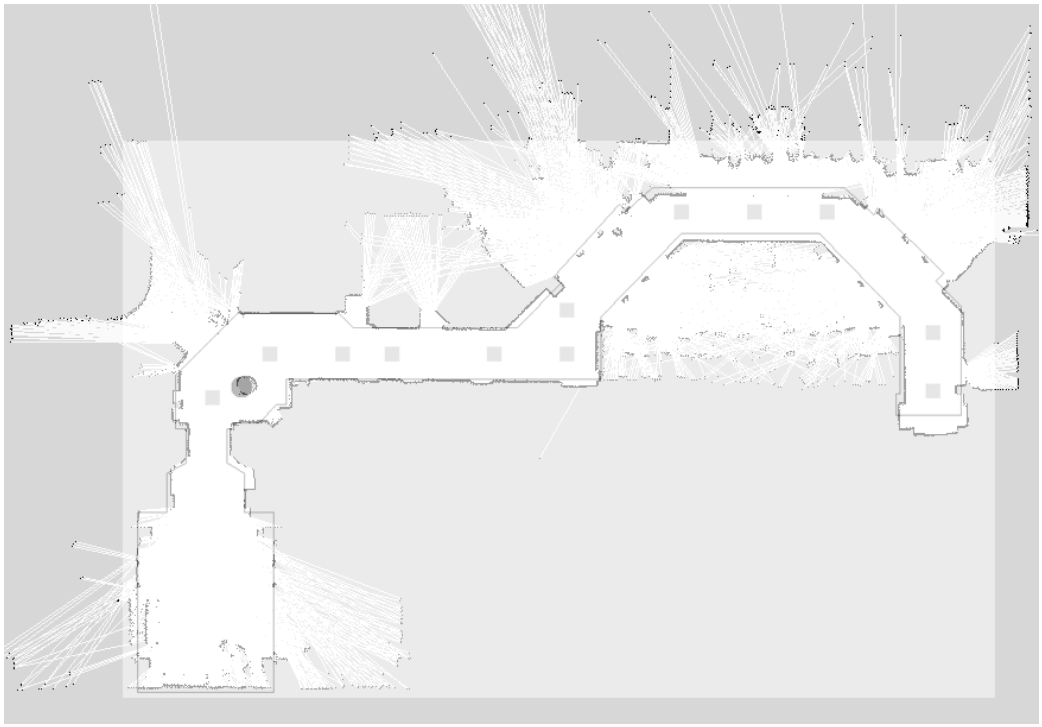

Figure 5.1.6: Digitized blueprints overlaid on map generated by GMapping SLAM

The problem of obstacles not decaying over time was determined to not pose a serious problem at the completion of this project. However, it is a feature that should be considered in the future.

5.2 Planning and Steering

For path planning, Roberto uses the standard ROS navigation stack. This system utilizes two planners: a global planner that creates a basic path to get from the initial position to the goal, and a local planner that generates steering commands and prevents the robot from hitting obstacles it can detect.

The global planner in the ROS navigation stack is "navfn". It takes the cost-map build by costmap_2d and applies Dijkstra's shortest path algorithm [31] to it with the robot's position as the start and the current goal position as the end. This

simple module generates optimal shortest paths from robot to goal while avoiding obstacles. It has some shortcomings. It is not useful for precision navigation, and it is not a dynamic planning algorithm. If the environment changes, it must re-plan its path. In the environment in which Roberto gives tours, there is fortunately enough space that this algorithm is useful without the need for precision navigation. Unfortunately, there are dynamic obstacles in this environment: people. In practice though, the planner does not take a noticeable amount of time to re-plan if someone crosses Roberto's path.

The local planner for the ROS navigation stack is "base_local_planner". The purpose of this module is to provide steering commands (angular and translational velocity) that the robot can follow. These commands should safely take the robot to its goal without colliding into any obstacles. This planner is configured on Roberto to use an algorithm known as Trajectory Rollout [27, 49].

The Trajectory Rollout finds steering commands by first generating sample steering commands. It will only generate commands that follow the robot's dynamic constraints and its current state. These constraints are specified in configuration files. These samples are forward simulated for a short amount of time. Each simulated command is one possible trajectory the robot can take. Each trajectory is scored with the cost function (5.2.1).

$$k_{path} * pathDist + k_{goal} * goalDist + k_{obs} * obs \qquad (5.2.1)$$

The best (lowset) scoring trajectory is what is commanded to the robot. This is repeated until the robot reaches the goal. In the cost function there are three k terms that serve as weights for the three inputs to the function. *pathDist* is the distance between the end of the trajectory (where the robot will end up if it follows this command) and the closest point on the path given by the global path planner. *goalDist* is the distance from the end of the trajectory to the goal. *obs* is the maximum cost for all points along the trajectory. The k weights used in this function are set in a configuration file. Figure 5.2.2 is a visual representation of the components of this cost function and their weighted sum.

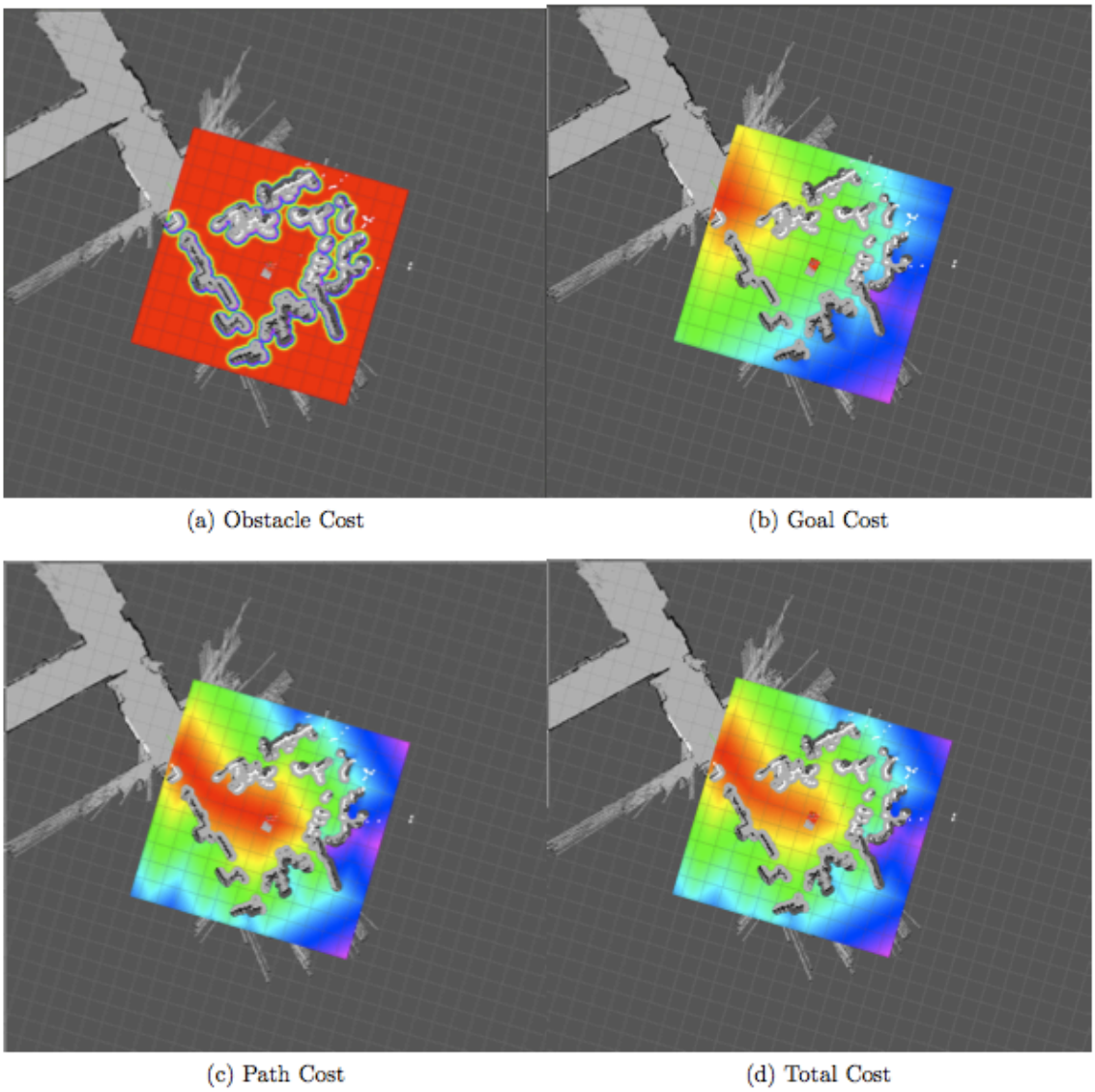

Figure 5.2.2: Cost Maps of showing the process of determining the best path through an environment.

There were other planners considered for Roberto. One method suggested was to have a pre-set track of virtual rails that the robot would follow using very tightly tuned steering. If something were blocking its path it would slow to a stop and first ask the object to move. Then if the obstacle were determined to be static, it would create a separate track that would take it around the obstacle and bring it back to the main tour track. This planner was abandoned because for this platform

it was better to have a planner that could handle arbitrary environments rather than specializing it to the sole task of traversing the hallway of Glennan.

Several experimental planners were also available for ROS. The SBPL (search-based planning) algorithms package contained several that were considered. Several were theoretically better planners than Trajectory Rollout. However, the software was in beta and both unstable and had not been reviewed. While building Roberto, mature stable technology was always chosen over new experimental and potentially dangerous alternatives.

All of these algorithms combine planning and steering. This means they must update fast enough to provide steering commands at a safe rate. These steering commands are converted to motor commands on the cRIO and fed as inputs into the PID for each motor. There are other planner steering paradigms where a path-planning module provides path primitives to a steering module. The advantage of this paradigm is slower planners can be used while the steering module runs asynchronously. A drawback to this approach is dynamic obstacle avoidance can be difficult to implement. It is the steering module's job to strictly follow the path provided. Since the steering module is unaware of possible obstacles, the planner must constantly be providing new correct paths to ensure a collision never occurs.

Chapter 6: Human Interaction

Attractive and functional human interaction is important for a successful tour-guide robot. The tourists need to be engaged in the experience. If the robot seems broken, dumb or defective, it can take away the magic of having a robot talking to you and showing off a location. Keeping this in mind, Roberto was designed to have limited but interesting human interaction with room for expansion. It was important not to set unrealistic goals in this aspect of the project. Failure here would result in unimpressed tourists and an unsuccessful tour. Roberto follows a scripted set of movements and speech passages as it goes through the tour.

6. 1 Speech Synthesis

There are a multitude of methods to generate speech on a computer, and many different tools and libraries exist for this problem. The simplest way to generate speech for a robot is to record an actual person speaking the dialog, then have the robot play these recordings back on demand. While this is an easy and effective method, there are drawbacks to this approach. A robot with a clearly recorded human voice does not sound like a robot. People expect a robot to sound somewhat mechanical. A clear human's voice feels wrong. Also, finding a good voice actor to deliver reasonable sounding dialogue can be difficult. Finally, when new dialogue is to be added to the robot--or if it were desired for the robot to dynamically create dialogue--this approach would become impractical. Because of this, procedurally generated speech techniques were evaluated for use on Roberto.

Many speech synthesizers were considered: Apple Say, Microsoft Text-to-Speech, gnuspeech, espeak, and Festival. Of these it was determined that Festival provided the best quality voice while being easy to use and highly configurable. Apple Say and Microsoft Text-to-Speech both provided very high quality voices but the voice clips would need to be prerecorded as sound files from these synthesizers so they could be used on Roberto's Ubuntu system. Looking forward, it would be impossible to dynamically generate speech with this approach. This was unacceptable because an important goal moving forward with the project is the ability to react to questions and respond uniquely to stimuli without a script.

Festival is an open source, general, multi-lingual speech synthesis system. It was developed by Alan W. Black [45]. Many researchers are still working on and with Festival, and thus Roberto could reap the benefits of advancements made on the Festival project in the future. Currently, Festival is used to generate sound files, which are played back on the robot using VLC bindings in Python. VLC [51] is an open-source multimedia player and library. While Roberto currently does not use any speech recognition software, other lab researchers have had success with the Sphinx [50] speech recognition system, and this is a feature planned for Roberto in the future.

6. 2 High-Level Controller

Roberto's high-level controller is responsible for selecting goal locations, determining when a goal has been reached, and triggering speech commands. It is a lightweight configurable script that functions as a state machine. The script reads in

a YAML file, which specifies locations and speech files. The controller steps through this list of locations (in *a priori* map space) and speech files, waiting to reach the next goal before starting the next command. This allows the platform to be used in many environments with only the need to change the YAML file and to generate new Gmapping localization and obstacle maps.

This controller is intentionally simple so it is easy to work with for both robotics researchers and researchers interested in changing the content of the tour. Its modular design allows for it to be easily replaced by a more complicated controller. There are plans to, in the future, incorporate a controller that includes emotional states and more complex dialog.

6. 3 Thermal Imaging Display

Roberto's display used for the thermal camera originally was intended to display an interactive avatar. There was interest in creating an emotive face that would animate the robot's voice and display an emotional state to make it more lifelike. Unfortunately, the computer selected for the platform could not support multiple monitors. However, this project had access to a thermal camera, which offered an interesting display. A thermal camera was mounted pointing behind the robot to capture the following tour group. The resulting thermal image is displayed on the Lilliput display. This display serves as an interesting attraction, allowing people to observe visualized heat signatures on their clothing and faces directly. During tours, families and children commented on and experimented with the display, helping to fulfill Roberto's entertainment objective. Although the thermal

camera is currently only used for a display, it offers opportunities for interesting future research and applications.

Chapter 7: Conclusions and Future Work

At the conclusion of this work, Roberto proved to be successful both as a versatile research platform and an entertaining tour guide. This was achieved by analyzing previous successful platforms and leveraging designs and goals from them. This also would not have been possible without leveraging the mature stable algorithms of ROS and the guidance of ROS's open-source community.

Although ROS proved very helpful, it was not without its shortcomings. Specifying *a priori* obstacles and areas the robot should avoid was necessary for this project, and ROS did not support this functionality. To overcome this shortcoming, a novel approach was introduced. A virtual sensor was created, which broadcast data that emulated detection of *a priori* specified obstacles. This virtual sensor provided a point cloud in a slice plane parallel to, but offset from the LIDAR sensor's slice plane. ROS interpreted this data as equivalent to a second LIDAR and thus incorporated the prespecified obstacles as though they were physically detected. This accomplished the goal of preserving prespecified (typically undetectable) obstacles.

An objective of this thesis was to make Roberto easy to use, modular, robust, safe, and stable. Roberto has proven to be safe and stable in approximately one year of use giving tours (although total running time is still low). There have been no injuries, accidents, or crashes in its operation since the completion of this work.

Subsequent to the work reported here, another researcher successfully added a Microsoft Kinect sensor without needing to alter the base software, demonstrating the success of modularity. Yet another researcher modified Roberto

to give tours in a new area at CWRU—the temporary home of Think[Box]. The process of adapting the robot to a new environment took approximately 3 hours: one hour to map the environment, and two more to create and debug the tour (which involved defining a new path and verbal script). This process used the Gmapping (SLAM) module first, to generate a localization map for the environment. Subsequently, AMCL was used to localize in the environment. Success in this effort by a new researcher in such a brief time validates the reusability of the base code and the ease of learning the software environment of Roberto to create novel variations.

In conclusion, Roberto demonstrated meeting the project goals. It has been well received as a capable tour guide, as evidenced by video recordings of visitors on tours. It has proven to be safe and reliable. It has been demonstrated to accept additions (a new sensor) easily, and it has been shown to be modifiable with relative ease create new tours. Research projects continue to be conducted using Roberto. It is a valuable platform that will serve the university for years to come.

While Roberto is a functioning, stable, mobile-robot research platform, the following are suggestions for future improvements

- Voice recognition: stable and reliable voice recognition should be developed for and implemented on the platform.
- Non-Tour Modes: there are many events at the university at which Roberto could server. Adding other interactive modes than the current tour mode would facilitate displaying it at these events.

- Thermal tracking: the thermal camera should have its signal fed into the Intel PC to allow for research on thermal marker tracking.
- Visual Slam: after the initial work in visual localization completed in this project, a stronger attempt to utilize visual localization with this platform should be undertaken.
- LIDAR-less operation; the LIDAR is the most expensive sensor on the robot. Future robot platforms may be created without this sensor due to budget constraints. Creating a stable platform that can localize and avoid obstacles without this sensor would be useful for compatibility with these robots.
- Wireless communication with TV monitors: during tour-guide operations, it is desired that the robot trigger useful information to be displayed on TV monitors along the tour path throughout the building.
- Automatic docking for charging: previous robotic tour guides have accomplished this with environmental modifications [23]. It is desirable to have the robot automatically charge itself and give tours unaccompanied in the future.
- Incorporation of sonar data and adding decaying data: the sonar and LIDAR data should be handled better for our purposes in ROS. A decay factor for LIDAR data should be added such that if data is older than a threshold, the planner deletes it. Because Roberto only has a few sonar sensors, conventional probabilistic sonar occupancy grid algorithms are not viable. The planner should only consider recent

sonar values. These additions may require considerable modifications to the ROS codebase and should be undertaken carefully with ample testing.

- Smoother motion: Roberto's motion can be jerky at times--especially when starting and stopping movement. This should be smoothed in future work.

Finally, the original mapping work begun within this thesis showed remarkable speed. It may be possible to incorporate this mapping computational approach into a ROS-compatible module. This would be a significant effort but could also be a significant contribution to ROS.

Bibliography

[1] Astigarraga Pagoaga, A.; Agent-based control architecture for a mobile robot, M.Sc. thesis for Dept. of Informatics, Universidad del País Vasco, 2001.

[2] Bishop G.; Welch. G.; An Introduction to the Kalman Filter. SIGGRAPH 2001 Course Notes, 2001.

[3] Brooks, R.A.; A robust layered control system for a mobile robot, IEEE J. Rob. Autom. 2 (1) (1986).

[4] Clarkson, M.R.; Sobel, A.E.K.;"Formal Methods Application: An Empirical Tale of Software Development", IEEE Trans. On Software Engineering, March 2002, vol 28, no. 3, pp. 308-320.

[5] Davison A. J., Reid I. D., Molton N. D., and Stasse O., "Monoslam: Real-time single camera slam," IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[6] Elfes, A. ; "Sonar-based real-world mapping and navigation," IEEE J. Robotics Automat. , vol. RA-3, no. 3, pp. 249-265, 1987.

[7] Fox, D.; Burgard, W.; Dellaert, F.; Thrun, S.; Monte carlo localization: Efficient position estimation for mobile robots. In Proceedings ofthe National Conference on Artificial Intelligence (AAAI), Orlando, FL, 1999. AAAI.

[8] Gat, E; Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In Proc. 10th National Conf. on Artificial Intelligence (AAAI-92). AAAI/MIT Press, 1992.

[9] Grisetti, G.; Stachniss, C.; Burgard, W.; Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics, 2006

[10] Hwang, S. Y., and Song, J.-B. (2008). Upward monocular camera based SLAM using corner and door features. Proc. of the 17th world congress of IFAC, 1663-1668.

[11] Jeong, W.; Lee, K. M.; "CV-SLAM: A new Ceiling Vision-based SLAM technique," in International Conference on Intelligent Robots and Systems, pp. 3070–3075, 2005.

[12] Kamon, I., and Rivlin, E. 1997. Sensory-based motion planning with global proofs. IEEE Transactions on Robotics and Automation 13(6): 814–822.

[13] Kanayama, Y.J. and Fahroo, F., "A New Line Tracking Method for Nonholonomic Vehicles," Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, NM, April 1997, p2908-2913.

[14] Kim G., Chung W., Kim, K. rock; Kim, M.; "The Autonomous Tour-Guide Robot Jinny," in In Proceedings of the IEEE/RSJ Inter- national Conference on Intelligent Robots and Systems (IROS), 2004, pp. 3450–3455.

[15] Lazea, Gh., Rusu, R.B., Robotin, R., Sime, R. (2006). The ZeeRO mobile robot - A modular architecture, RAAD 2006 International Workshop on Robotics, ISBN 963-7154-48-5, Balaton, Hungary

[16] Lemaire T., Berger C., Jung I., and Lacroix S., "Vision-based SLAM:Stereo and monocular approaches," Int. J. Comput. Vis., vol. 74, no. 3, pp. 343–364, 2007.

[17] Low, K. H.; Leow, W. K.; and Ang, Jr., M. H. 2002. A hybrid mobile robot architecture with integrated planning and control. In Proc. AAMAS, 219–226.

[18] Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B.; Konolige, K.; "The office marathon: Robust navigation in an indoor office environment," in Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), 2010.

[19] Mataric, M.J., (1992), "Behavior-Based Systems: Key Properties and Implications", in IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France, pp.46-54.

[20] Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit. B.; "Fast-SLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in Proc. Int. Joint Conf. Artif. Intell., 2003, pp. 1151–1156.

[21] Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit. B.; FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In AAAI, 2002.

[22] Newman W.; Covitch A.; May R.; A client/server approach to openarchitecture, behavior-based robot programming. Space Mission Challenges for

Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on, page 8, 2006.

[23] Nourbakhsh, I.R.; Bobenage, J; Grange, S; Lutz, R; Meyer, R; Soto, A; "An Affective Mobile Educator with a Full-time Job," Artificial Intelligence, Vol. 114, No. 1 - 2, October, 1999, pp. 95 - 124.

[24] Nourbakhsh, I.R.; The failures of a self-reliant tour robot with no planner. See http://www.cs.cmu.edu/illah/SAGE/index.html, 1998.

[25] Nourbakhsh, I. R.; Kunz, C. ; Willeke, T.;"The mobot museum robot installations: A five year experiment," in In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2003, pp. 3636–3641.

[26] Lina María Paz, J. D. T., Piniés, P. and Neira, J. (2008). Largescale 6-DOF SLAM with stereo-in-hand. IEEE Transactions on Robotics, 24(5): 946–957

[27] Perko, E.; Newman, W.; "Precision Navigation for indoor Mobile Robots" Case Western Reserve University, EECS Dept. Masters Thesis, Forthcoming (anticipated summer 2012).

[28] Propkop, N.; Autonomous camera calibration for visual navigation. Case Western Reserve University, EECS Dept. Masters Thesis, May 2010

[29] Philippsen R., Siegwart R. Smooth and Efficient Obstacle Avoidance for a Tour Guide Robot. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2003.

[30] Reifel, S. The SRI Mobile Robot Testbed — A Preliminary Report, Technical Note 413. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Feb 1987.

[31] Russell S. and Norvig P.. Artificial Intelligence: a modern approach. Prentice-Hall, 2nd edition, 2003.

[32] Saffiotti, A.; and Ruspini, E.; and Konolige, K.; A Fuzzy Controller For Flakey, An Autonomous Mobile Robot, Technical Note 529. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1993.

[33] Saffiotti, A.; and Konolige, K.; and Ruspini, E.; A Multivalued Logic Approach To Integrating Planning Control, Technical Note 533. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Jun 1993.

[34] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso, "Xavier: Experience with a layered robot architecture," ACM magazine Intelligence, 1997.

[35] Simpson, J.; Jacobsen, C.; Jadud, M.; Mobile Robot Control: The Subsumption Architecture and occam-pi. In Frederick R. M. Barnes, Jon M. Kerridge, and Peter H. Welch, editors, Communicating Process Architectures 2006, pages 225–236. IOS Press, September 2006.

[36] Smith P., Reid I., and Davison A., "Real-time monocular SLAM with straight lines," in Proc. Brit. Mach. Vis. Conf., 2006, vol. 1, pp. 17–26.

[37] Sunderhauf, N.; Konolige, K.; Lacroix, S.; Protzel, P.; "Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle," in

Tagungsband Autonome Mobile Systeme. New York: Springer-Verlag, 2005.

[38] THIERFELDER, Susanne ; SEIB, Viktor ; LANG, Dagmar ; HÄSELICH, Marcel; PELLENZ, Johannes ; PAULUS, Dietrich: Robbie: A Message-based Robot Architecture for Autonomous Mobile Systems. In: INFORMATIK 2011, 2011

[39] Thrapp, R.; Westbrook, C.; Subramanian. D.; Robust localization algorithms for an autonomous campus tour guide. In Proc. of the IEEE International Conference on Robotics & Automation, 2001.

[40] Thrun, S.; Bennewitz, M.; Burgard, W.; Cremers, A.; Dellaert, F.; Fox, D.; Hahnel, D.; Rosenberg, C.; Roy, N.; Schulte, J.; and Schulz, D. 1999. MINERVA: A second generation mobile tour- guide robot. Proc. of ICRA-99.

[41] Thrun, S.; A. Buecken, W. Burgard, Dieter Fox, T. Froehlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt, "Map Learning and High-Speed Navigation in RHINO," AI-based Mobile Robots: Case Studies of Successful Robot Systems, Kortenkamp, D. and Bonasso, R.P. and Murphy, R., ed., MIT Press, 1998

[42] Thrun, S.; Burgard, W.; Fox, D.; Intelligent Robotics and Autonomous Agents, MIT Press, 2005

[43] Utz, H. ; Sablatn¨og, S.; Enderle, S.; and G. Kraetzschmar. Miro - middleware for mobile robot applications. IEEE Transactions on Robotics and Automation, 18(4):493–7, August 2002.

[44] Wilkins, David E. High-Level Planning In A Mobile Robot Domain, Technical Note 388. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Jul 1986.

[45] The Festival Speech Synthesis System, 2012, <http://www.cstr.ed.ac.uk/projects/festival/>

[46] Intelligent Ground Vehicle Competition website 2012, <http://www.igvc.org/>

[47] OpenRDK Homepage, 2010, <http://openrdk.sourceforge.net/>

[48] The Orocos Project Homepage, 2012, < http://www.orocos.org/>

[49] ROS.org documentation wiki, 2012, <http://www.ros.org/wiki/> [B] Dimension Engineering Sabertooth 2x25 spec sheet, 2012, <http://www.dimensionengineering.com/datasheets/Sabertooth2x25v2.pdf>

[50] Sphinx website , 2012, <http://cmusphinx.sourceforge.net/>

[51] Video LAN Organization website, 2012, <http://www.videolan.org/vlc/index.html>

[52] Ferrick, Allen; Fish, Jesse; Venator, Edward; Lee, Gregory S.; , "UAV obstacle avoidance using image processing techniques," Technologies for Practical Robot Applications (TePRA), 2012 IEEE International Conference on , vol., no., pp.73-78, 23-24 April 2012