NAVIGATIONAL SENSING FOR THE EDUBOT

NSF Summer Undergraduate Fellowship in Sensor Technologies
Sriram Radhakrishnan (Electrical & Systems Engineering) – University of Pennsylvania
Advisors: Dr. Daniel E. Koditschek[1], Dr. Galen Clark Haynes[2]

Summer 2010

ABSTRACT

Automation – making a robot perform human functions but without human control – is the overarching goal of robotics. An important aspect of autonomous robots is the ability to detect and avoid obstacles. The goal of this project was to determine the feasibility of enabling a hexapedal robot, known as the EduBot, to generate maps of its surrounding area. This was accomplished by equipping the EduBot with a laser scanner, which was then controlled by the ROS robotic software package. The laser scanner was used to measure the distances to obstacles surrounding the EduBot. These measurements were then used by a ROS program to generate a map of the environment. The EduBot mapped various areas within a building to provide a large sample for analysis. These maps were then compared with the actual area to determine their accuracy. Once software and hardware errors discovered in early maps were corrected, the EduBot was able to generate accurate representations of its surrounding environment, thus allowing the EduBot to sense it surroundings. This suggest that autonomous navigation by the EduBot is possible.

[1] Professor, Electrical & Systems Engineering, University of Pennsylvania
[2] Post-Doctoral Fellow, Electrical & Systems Engineering, University of Pennsylvania

Table of Contents

1. INTRODUCTION

Every day, humans avoid obstacles as we navigate through the world. As this process is done subconsciously, we do not realize the complex calculations that are required to perform this task. Humans use sensory data from various organs to figure out a safe path between two points. This path is then updated as new obstacles are sensed.

As with humans, robots must be able to avoid obstacles. As robots slowly become more autonomous, they need to be able to deal with the physical objects in their surrounding area. [1] For example, when someone commands a robot to go through a dangerous building in search of something, that person does not know what is inside the building and what areas are navigable; the robot needs to figure that out for itself. In order for robots to perform the same task, they would need to have similar data, provided by sensors mounted on the robot. [1] Each sensor would measure something about the surrounding environment.

However, as Cmdr. H.R. Everett (U.S. Navy) writes, "There has been quite a tendency in many cases to oversimplify these issues, and assume that the natural growth of technology will provide the needed solutions." [1] In other words, engineers have not devoted a great amount of time towards developing solutions for obstacle avoidance, something this project aims to do.

One of the most important senses is that of sight; to a robot, sight is essential to detecting obstacles. A robot sees by using a sensor to map the layout of its surrounding environment. This project deals with implementing a laser scanner on a legged robot to enable sight-based navigation. In addition, in order to simplify the software implementation of the laser scanner, a new robot software package was used, known as ROS (Robot Operating System). ROS was chosen because it contains many built-in programs that were useful in this project *(see Section 2.3)*. [2]

The Background section *(Section 2)* explains some of the concepts used in this project in order to give a better understanding of the procedures and results discussed in later sections. In addition, each subsection briefly explains the reasoning behind using that particular concept in this project.

Sections 3 and 4 present the methods used and the results gained from this project, as well as a discussion of these results. Sections 5 and 6 summarize the findings from this project, their consequences, and recommendations for future exploration of navigational sensing on the EduBot.

2. BACKGROUND

There are four main things necessary to implement an obstacle avoidance system in a robot: a robot, an object sensor, a software package that can control both the robot and the sensor, and positioning. Each of these items is described below.

2.1. The EduBot and the RHex Robotic Platform

RHex is a family of six-legged robots designed to travel on multiple terrains. [3] The robots use legs rather than wheels because legs are better able to adapt to changes in terrain. [4] The intent of the RHex project is to create a biologically-inspired robot that can be used in many applications, particularly in outside environments. [3] One of the many adaptations of the RHex platform is the EduBot *(shown in Figure 1)*, which is primarily used for education and research. [3] This project aims to augment the capabilities of the EduBot by adding a laser scanner that can aid in autonomous navigation.

Figure 1: EduBot
(Source: KodLab Website, kodlab.seas.upenn.edu)

2.2. Light Detection and Ranging (LIDAR)

The key to avoiding obstructions is being able to sense them. This requires the use of a variety of sensors. These sensors must not only provide good data but must also work with the limited resources (e.g. power, space) available on a mobile robot. [1] Researchers are most interested in non-contact detection methods – ways of finding obstacles without actually touching them. [1] The most common sensors use either sonar or laser, each having its advantages and disadvantages. [1] A laser range finder has been selected for this project.

Light Detection and Ranging, or LIDAR, is a process whereby a laser beam is used to determine various characteristics of the surrounding environment. [5] A laser beam is aimed at a point in the world. The reflection of this beam is then sensed by a receiver. By comparing the transmitted beam with the returned beam, various characteristics can be measured of the point in space, including distance. Since a laser beam is pure in that it is composed of very few wavelengths, it performs well in sensing.

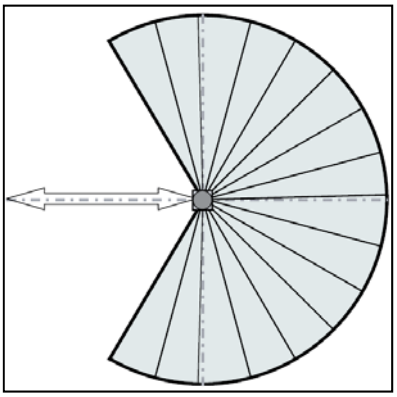

Figure 2: Hokuyo URG-04LX-F01 Laser Scanner
(Source: Hokuyo Automatic Co.)

The laser scanner used in this project contains a laser mounted on a rotating platform. By spinning this platform at high speeds, the laser scanner measures the distance to the nearest obstacle at all points in the angular range, as illustrated in Figure 2. The center point represents the laser scanner, while the shaded area shows the measurable area. This measureable area covers a range of 240^{o} around the laser scanner.

2.3. Robot Operating System (ROS)

The Robot Operating System, commonly known as ROS, is an open-source software package created for use on a variety of robotic platforms. [6] Being open-source, ROS is freely available for anyone to edit. This has led to the creation of numerous supplemental programs that add to ROS's functionality, many of which are used in this project. [2] ROS employs modular programming, a concept in which each part of a robot is controlled by a different program (called a node). [6] This way, it is easier to implement new features and add them to the existing ROS framework. This concept also simplifies the integration of ROS with the EduBot's existing software framework, and thus has been chosen for this project.

ROS employs a language-neutral interface for communication. [6] This interface consists of messages whose format is defined in a short text file listing the fields of the message. These messages are posted to topics, which are essentially message boards for ROS nodes. [2] Each topic contains information that a program has posted to it. For example, in this project, a laser scanner posts data gathered (i.e. distance measurements) to the *base_scan* topic. This use of messages and topics allows ROS to be, in theory, language neutral. This means that programs can be written in a variety of languages as each program is self-contained. All connections between nodes are handled through the language-blind message system, eliminating the need for programs to communicate with each other directly. In practice, four main programming languages are supported – C++, Python, Lisp, and Octave. [6] This project primarily uses Python, with a few programs written in C++.

The various programs in ROS are organized into stacks and packages. A package is a collection of programs and other code that work together to perform a certain task. [2] A stack, similarly, is a collection of packages that achieve a certain goal. [6] This project uses several ROS packages, such as the *gmapping* package for generating maps. The base ROS structure includes several tools to help with navigating through the ROS file structure, as well as error debugging and status checking. Since nodes in ROS are connected in a graph-like structure (i.e. with nodes connected together like a tree), one ROS tool, called *rxgraph*, displays a visual representation of the present ROS structure, including all running nodes and topics. [6] An example of this tool, adapted from the ROS tutorials, is shown in Figure 3.

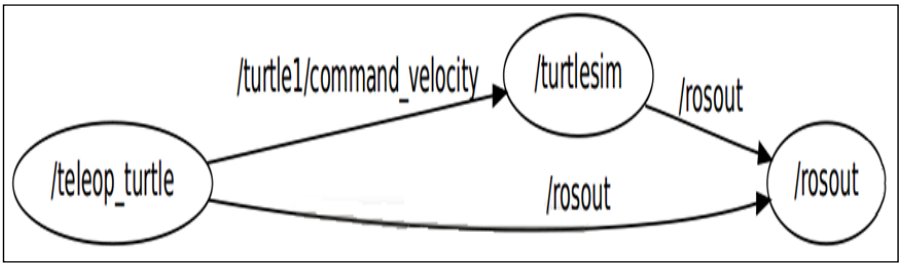

Figure 3: Example of the Graphical Representation of the ROS System Structure
(Source: ROS Tutorials, www.ros.org/wiki)

Each oval represents a ROS node, while each arrow represents a communication topic. The terminology explained in this section is used in later sections when discussing this project.

2.4. Positioning

In order to generate a map of its surroundings, a robot must be able to match each laser distance measurement with a point in space, typically denoted by using a traditional Cartesian coordinate system and a rotation angle. There are many methods of determining a robot's position, such as odometry and global positioning systems. [7] Because of its simplicity and relative ease of use, odometry has been chosen for this project *(see Section 2.4.1)*. In addition, the ROS mapping utility used to generate the map employs a method known as Simultaneous Localization and Mapping (SLAM) to correct some of the error inherent with odometry *(see Section 2.4.2)*.

2.4.1. Odometry

Odometry is a method of determining a robot's position relative to its starting point. By knowing, for example, the circumference of its wheel or its speed, a robot can be programmed to calculate how far it has moved. [7] In fact, odometry is the most commonly used method for robot positioning, as it is relatively simple and provides usable data for a low cost. [7] However, since odometry is based on the iterative summation of robot movements, a great deal of error occurs over time. [7] This error can be caused both by systematic discrepancies, such as an incorrect wheel circumference, and by non-systematic discrepancies, such as wheel slippage. While systematic error can be corrected by calibration, non-systematic error is much more difficult to fix. Therefore, while odometry allows for a general position, it cannot provide a specific position with only negligible error. For this project, however, the accuracy of odometry is sufficient, particularly as the ROS program used for mapping employs a process known as SLAM *(See Section 2.4.2).*

2.4.2. Simultaneous Localization and Mapping (SLAM)

Simultaneous localization and mapping (SLAM) is a technique to compute the current position of a robot relative to its starting position while generating a map of its environment. [8][9] As Sebastian Thrun writes, "The SLAM problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robots." [8] The SLAM algorithm is based on two things: knowing the starting location of the robot, and being able to sense the surrounding environment. In this project, the starting location is assumed to be the origin, or (0,0). Sensing the surroundings of a robot is the primary goal of this project. The SLAM process relies on generating a map of the robot's surroundings and then figuring out the position of the robot using this map. As the robot moves, it gathers data regarding its surroundings from a sensor, which in this project was the laser scanner. The SLAM method compares successive sets of laser data (i.e. the laser view at point A with the view at point B), and analyzes the differences between them. For example, if the robot moved forward between points A and B, then the image of the environment at point B would be very similar to the image at point A, but the distances would be different to account for the robot's movements. Using these changes, SLAM can compute the change in the robot's position. Since each image is essentially a map, this process is also known as map matching. [7] The ROS *gmapping* program uses a combination of SLAM computation and odometry information from the robot to determine the robot's position relative to its starting point. [2] While SLAM is fairly accurate, it can fail if there is not enough laser data to compute a position change. In these instances, *gmapping* uses odometry to determine the robot's position.

3. EXPERIMENTAL PROCEDURES

The laser scanner used in this project was the Hokuyo URG-04LX-F01 Scanning Laser Range Finder *(shown in Figure 4)*. This laser scanner employs a Class 1 laser mounted on a spinning base, enabling scanning in a range of 240^{o} around the device. [10] It is powered by a 12VDC input.

Figure 4: Hokuyo URG-04LX-F01 Laser Scanner
(Source: Hokuyo Automatic Co., Ltd.)

To generate a map, the ROS gmapping utility requires two transformations between coordinate frames: *laser-to-robot* and *robot-to-odom*, where *odom* is short for odometry. The former accounts for the difference in position between the center of the laser scanner and the center of the robot, while the latter is the position of the robot determined using odometry. These transformations are recorded in ROS using the *tf* package, which contains code for publishing transforms to the *tf* topic using a uniform format.

In order to learn ROS and how the laser scanner worked, both were implemented using a wheeled cart and a laptop (to mimic a robot) before being implemented on the EduBot. Each of these implementations is explained in the subsections below.

3.1. Cart Implementation

A Lenovo ThinkPad T510 Laptop was used as the computational unit. The laptop was running Ubuntu Linux 10.04, on which ROS was installed. The laser was then connected to the laptop via USB, using drivers provided by the manufacturer. To emulate the EduBot, the laptop and the laser scanner were placed on a wheeled cart. The laser scanner was powered using a standard AC power adapter connected to an extension cord.

In the cart implementation, the *laser-to-cart* transformation was programmed with no shift, as the difference in position was not essential to accurate map building. For the odometry data, a Bash script was written so that the distance traveled could be easily input into the program. The script accepted changes in position using the arrow keys, each representing the appropriate direction (e.g. up key for forward, left key for left). When the script detected a key-press, it would update a ROS topic (named "position") with the new coordinates. Each key-press denoted a 0.1 meter shift in the given direction. To convert these coordinates to the appropriate transformation format, another ROS program was written that subscribed to the *position* topic, took the coordinate information, and published the transform using commands from the *tf* package. The script was written using the Bash scripting language, while the conversion node was written using Python. The program for the static *laser-to-cart* transform was written in C++.

The laser was connected to ROS using the *hokuyo_node* package, a built-in ROS interface for Hokuyo laser scanners. This package contains a ROS node, also called *hokuyo_node*, which converts raw laser data into ROS messages posted on the *base_scan* topic.

The cart containing the laser and the laptop was then taken to various locations within the University of Pennsylvania's engineering buildings. At each location, the transform and laser data were recorded (i.e. saved to a file) while the cart was moved around a small area. In each instance, the cart was moved along walls until the entire area was mapped. To ensure that the area was mapped in its entirety, the cart was moved in a loop.

After the first few runs, the *gmapping* utility was used to generate the maps. Generating the maps involved playing back the recorded laser and transform data while running the *gmapping* ROS node. These initial maps were not very representative of the measured area *(see Figure 7 in the Results section)*. To correct this error, settings regarding the accuracy of the odometry information were changed. These values, named *srr, srt, str, stt*, essentially told the *gmapping* utility how much to trust the odometry information, and were reported on a scale of 0.0 to 1.0, with 1.0 representing very poor accuracy. However, setting all four of these values to 0.9 did not considerably change the map *(see Figure 8 in the Results section)*. A review of the code and of the gmapping utility found that the angular position of the cart was not input correctly. Initially, the angular position of the cart (in the *cart-to-odom* transform) was set to 0 radians. As in the standard angular coordinate system, 0 radians represented the positive x-axis direction. However, the inputs to the odometry script were based on the fact that the cart was actually pointing in the +y-axis direction, or $\pi/2$ radians. The *cart-to-odom* transform was then updated with the correct angle. The resulting maps were much more representative of the measured area *(see Figure 9 in the Results section)*.

Initially, the manual odometry script only accepted inputs of forward, backward, left, and right. However, the EduBot is not a *holonomic* robot, meaning that it cannot move sideways; rather, it can only turn left or right. To account for this limitation, the script was updated to accept inputs for rotating left or right. Each time the appropriate key was pressed, the angular position would change by $\pi/4$ radians. This change enabled the cart to better mimic the EduBot, as well as to generate more complete maps *(see Figure 11 in the Results section)*.

3.2. EduBot Implementation

After understanding the laser scanner, the relevant ROS programs, as well as ROS itself, we moved to the EduBot. All the experiments in this project were conducted on the EduBot known as *Penn3*. The EduBot was powered using a 14.8V, 1320mAh lithium-polymer battery. The EduBot was controlled by a built-in CPU and ran Ubuntu Linux 8.04. The laptop (Lenovo ThinkPad T510) was used to connect via SSH to the EduBot over a Wi-Fi connection. This connection was executed both through a Linksys access point as well as through an ad-hoc connection. The ad-hoc connection, which connected the laptop directly to the EduBot, later became necessary since mapping occurred in areas out of range of the access point.

To power the laser scanner on the mobile EduBot, the 12V port on the EduBot's battery management board was used. A connector was fashioned using a standard male AC adapter plug with two wires on the other side. These wires connected to the EduBot, while the adapter plug connected to the laser scanner. The laser did not greatly affect the battery life on the EduBot. The EduBot draws approximately 30W when walking and 10W when stopped, while the laser draws approximately 4.4W [10]. This additional power draw from the laser decreased the battery life by about 20%, according to calculations. While the calculated run time of the EduBot for pure walking was about forty minutes, the calculated run time with the laser on only decreased to about 33 minutes. In practice, battery life was not an issue, as each mapping run took less than fifteen minutes. In addition, to ensure sufficient voltage, each mapping run was executed on fully-charged batteries. The laser itself was mounted on the EduBot shell using cable ties and electrical tape *(see Figure 5)*.

Figure 5: EduBot with Mounted Laser Scanner

Due to limitations of the laser scanner driver used on the EduBot, the laser scanner was connected to the laptop using a USB cable as before. While this kept the laptop tethered to the robot, mobility was maintained by placing the laptop on a cart. In addition, ROS was unable to be installed on the EduBot since its on-board data storage space was insufficient. The EduBot used a 2GB compact flash card, while ROS itself requires more than 1GB. While a bigger card was tried, there were unsolvable issues with booting to the new card.

As with the cart implementation, a program was needed to compute the position of the EduBot using odometry. The EduBot uses a software package called Dynamism. Dynamism is a lightweight, real-time robot control environment used to control the EduBot platform. [11] Dynamism allows a computer to interface with the EduBot to acquire data, such as speed settings, using either Python or C++ commands. The speed of the EduBot is reported using an arbitrary scale with positive numbers incremented by one-tenth. A speed of 1.0 was chosen for its relative stability and steady movement. At

this setting, the actual speed, in cm/s, of the EduBot was measured. After taking several measurements in different environments, an average speed of 0.15 meters per second was found. While the EduBot can both turn-in-place and turn while moving, only turning-in-place was used, since the EduBot did not have a consistent angular speed when turning while moving. For turning-in-place, the EduBot has only two speed values: 1 and -1. Because the odometry would be calculated using time, the turning speed was measured in radians/sec, and was approximately $\pi/8$ radians per second. This was not a very accurate measurement however, and in practice, turning required some manual adjustments.

Using these speed measurements, an odometry node was programmed in Python for use with ROS. This node interfaced with Dynamism on the EduBot and queried Dynamism for the values of the speed and turn variables. Based on the value of each variable, the node adjusted the coordinates of the EduBot accordingly. Once this node was programmed, it was tested for accuracy. As soon as the program was run, however, the EduBot would crash, apparently because it was unable to handle both motor control and transmitting data to the computer. After some trial and error, the node was rewritten to use a different command to read in the variables. This command specified the time interval between variable reads, thus reducing the speed of reading. After setting this value at one second, the EduBot performed without any problems. This interval was then reduced to 0.5 seconds for better positional accuracy.

Another ROS node was programmed for the transformation between the laser frame and the EduBot frame. As stated in Section 3.1, this transformation was ignored in the cart implementation as it was not important. With the EduBot, however, this is crucial to ensure that the distance measurements are from the point of view of the EduBot. This difference in position was measured to be -0.04 meters in the x-axis and -0.11 meters in the z-axis. As the laser was mounted in the center of the robot, there was no change in the y-axis.

The EduBot was then taken to various areas to map the environment. The mapping process itself was fairly simple. Since odometry data was automatic and did not have to be entered manually as with the cart implementation, there was little user input necessary to acquire the odometry and laser data other than to control the robot. The robot was controlled using a Logitech joystick pad, with buttons for functions such as calibrate and stand, as well as control sticks for walking and turning.

The major problem found during mapping was with connectivity. At random times while mapping, the laptop would lose connectivity with the EduBot, thus requiring a full reset of the robot and starting the mapping process again. Initially, this was thought to be a problem with the CPU, in that the CPU would randomly reset itself. However, further examination revealed that the wireless card used for Wi-Fi connectivity on the EduBot was the cause of the problems. While removing the card and plugging it back in restored the connection, this was unfeasible as it required input using a keyboard and a monitor, which are unavailable when away from the lab. The problem was eventually attributed to poor USB ports on the EduBot. A new wireless card was attached to the

EduBot using a USB extender, which seemed to work better. In addition, two batteries were used instead of one to better maintain the voltage necessary to run the robot.

As the EduBot moved forward, it would turn slightly and not go straight. In order to easily correct this, the EduBot odometry program was updated to allow for minute turning adjustments while moving without changing the position angle. While this change made turning possible only when stopped, it was necessary to ensure a higher accuracy for the odometry data.

Because of the lesser accuracy of the EduBot odometry data compared with the odometry in the cart implementation, the quality factors (i.e. the parameters of the ROS *gmapping* program) were set at 0.7 for all the maps taken with the EduBot.

4. RESULTS

In the cart implementation, maps were taken in two main locations within the School of Engineering: Berger Auditorium Lobby in Skirkanich Hall (referred to as Berger) and the Graduate Research Wing 3rd Floor Elevator Lobby in the Moore Building (referred to as GRW). For comparison with the maps, a picture of the GRW area is shown in Figure 6, while the Berger area is shown in Figure 10. Initially, as stated above, the angular position was not set correctly. This resulted in a very poor representation *(see Figure 7)*. This poor quality was also not noticeably affected by changing the odometry accuracy settings in the *gmapping* program *(see Figure 8)*. Once the angular position was corrected, the resulting map was a much better representation of the actual area *(see Figure 9)*.

Figure 6: The Mapped GRW Area

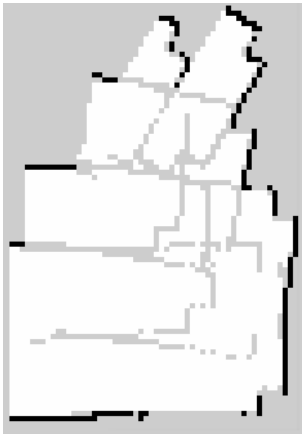

Figure 7: Map before any Correction (Location: GRW)

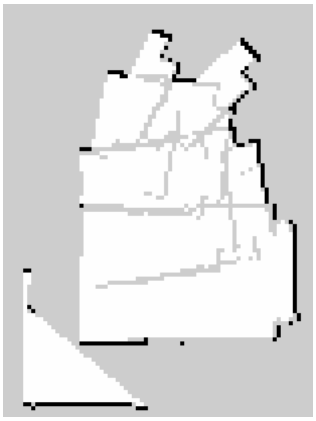

Figure 8: Map with Changed Accuracy Settings (Location: GRW)

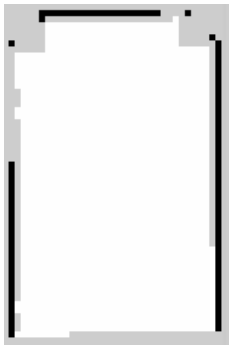

Figure 9: Map after Angular Position Fix (Location: GRW)

After the cart's odometry script worked for simple movements, the ability to rotate was added, resulting in the map shown in Figure 11. This map increased the accuracy of the map, since laser data were gathered for areas all around the cart, instead of just in the laser's measurable range *(see Figure 2)*.

Figure 10: The Mapped Berger Area

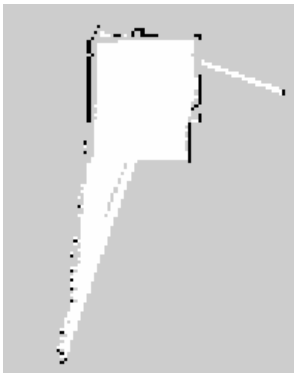

Figure 11: Map after Rotation Feature Added (Location: Berger)

After the cart implementation was successful, we moved to the EduBot. Because of various connectivity and power problems encountered *(see Section 3.2)*, each map took much more time to produce. However, because of the mobility of the EduBot, much larger areas were able to be mapped.

The primary mapping location for the EduBot was a hallway in the Towne Building *(shown in Figures 12, 13, and 14)*. A map of the area shown in Figure 12 was taken first *(see map in Figure 15)*, followed by the entire area shown in Figures 12, 13, and 14 *(see map in Figure 16)*.

Figure 12: Mapped Area in Towne Building Hallway (Part 1, front)

Figure 13: Mapped Area in Towne Building Hallway (Part 2, back-right)

Figure 14: Mapped Area in Towne Building Hallway (Part 3, back-right)

Figure 15: Map of Area in Figure 11

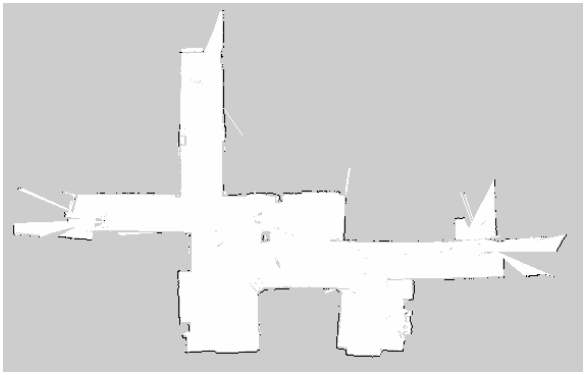

Figure 16: Map of Entire Towne Building Hallway from Figures 11, 12, and 13

As shown in Figures 15 and 16, the maps generated using the EduBot are a good representation of the actual area. The points where the map becomes inaccurate (e.g. on the top and bottom of Figure 15, as well as on the very left and right portions of Figure 16) are due to those areas not having any walls or other obstacles to sense. Thus, the laser scanner did not gather any data about those regions.

5. DISCUSSION & CONCLUSION

The purpose of this project was to determine the feasibility of using a laser scanner mounted on a robot to map out an area. Using a commercially available Hokuyo laser scanner, we mapped various indoor areas, first using a wheeled cart and then using the EduBot. Initially, these maps were very unrepresentative of the actual area, owing to various errors in programming, as shown in Figures 7 and 8. These errors were fixed in the cart implementation, resulting in good quality maps *(Figure 9)*. The EduBot implementation was more complex, as it required more programming to integrate the ROS programs with the EduBot's own software. We faced some issues with inaccurate odometry, as was expected from Everett, et al.'s statements [7]. However, the resulting maps were a relatively good representation of the mapped area *(Figures 15 and 16)*. These maps suggest that they can be used with programs such as the ROS stack navigation to allow for autonomous navigation.

6. RECOMMENDATIONS

Based on the outcomes of this project, further implementation of laser-based navigation can be attempted. ROS has a stack called *navigation* that uses transform and map data to navigate a robot to a specific pose. Since both transform and map data has been successfully gathered for the EduBot in this project, it may be possible to use the *navigation* stack with the EduBot. Before implementing, the maps should be more closely checked for accuracy to prevent the EduBot from crashing into obstacles. While the maps in this report were resized for space, in the original map files, each pixel represented 0.050 meters. Using this scale, comparisons can be made between distances from the map and measurements of the actual mapped areas. If these values match with negligible error, then the map can be deemed sufficiently accurate for use with autonomous navigation. While this method was used to check a few measurements on the maps, there was insufficient time to verify every distance. In addition to distance accuracy, taking several maps of the same area and comparing their relative accuracy would ensure that the mapping process can be repeated with equal accuracy.

There also might be ways to improve the odometry of the EduBot, both in the code and physically on the robot. The robot's gait was not perfectly straight and did slip, particularly during turns. Legs with better grip or a different gait may work better. Installing a sensor on the EduBot's legs to better compute distance traveled may also improve odometry. In addition, using a global positioning system will probably improve positional accuracy. Using a GPS would only be possible outdoors, but because of the relative unevenness of outdoor terrain, odometry would be even less reliable than indoors, thus making GPS a viable solution.

Another consideration is the amount of G-force felt by the laser. The laser scanner used in this project was specified to withstand only a certain amount of G-forces. While this was not a problem when testing in a flat indoor area, the EduBot may face strong forces when used in outdoor areas, such as a desert or a rocky terrain. Installing shock absorbers or in some way protecting the laser would be necessary to use the robot in such an environment.

Finally, while space limitations prevented installing ROS on the EduBot, this can easily be fixed by increasing the on-board disk space. Running ROS directly on the EduBot eliminates the need to tether the EduBot to the laptop for the laser, which would allow for easier mapping of large areas. This may also reduce the effect of connectivity issues between the EduBot and the laptop.

7. ACKNOWLEDGEMENTS

I would like to thank Professor Daniel E. Koditschek for giving me the opportunity to work in the KodLab this summer. I would also like to thank Dr. Galen Clark Haynes, Aaron Johnson, and B. Deniz Ilhan for all their help and advice throughout the course of my project. Finally, I would like to thank Professor Jan Van der Spiegel and the University of Pennsylvania for hosting and organizing the SUNFEST program, and the National Science Foundation for their continued financial support of the SUNFEST REU.

8. REFERENCES

[1] C. H. R. Everett, "Survey of collision avoidance and ranging sensors for mobile robots," *Robotics and Autonomous Systems,* vol. 5, pp. 5-67, 5, 1989.

[2] Willow Garage, "ROS Wiki," vol. 2010, 10/1/2009, 2009.

[3] KodLab. 2010, KodLab : Rhex. *2010(6/23),* Available: http://kodlab.seas.upenn.edu/RHex/Home.

[4] U. Saranli, M. Buehler and D. E. Koditschek, "RHex: A Simple and Highly Mobile Hexapod Robot," *The International Journal of Robotics Research,* vol. 20, pp. 616-631, July 01, .

[5] J. B. Campbell. 2007, "LIDAR," in *Introduction to Remote Sensin*g (4th ed.) Available: http://books.google.com/books?hl=en&lr=&id=r1JwIxMGYUAC&oi=fnd&pg=PR15&dq=LIDAR+introduction&ots=rT8SVk933M&sig=PtDQicu_ULkixyxMaRo0z5pVIiI#v=onepage&q=LIDAR%20introduction&f=false.

[6] M. Quigley, B. Gerkey and et al. ROS: An open-source robot operating system. Available: http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf.

[7] J. Borenstein, H. R. Everett, L. Feng and D. Wehe, "Mobile robot positioning: Sensors and techniques," *J. Robot. Syst.,* vol. 14, pp. 231-249, 1997.

[8] S. Thrun, "Simultaneous Localization and Mapping," *Robotics and Cognitive Approaches to Spatial Mapping,* pp. 13-41, 2008.

[9] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *Robotics & Automation Magazine, IEEE,* vol. 13, pp. 99-110, 2006.

[10] L. Hokuyo Automatic Co., "Scanning Laser Range Finder Specifications," vol. 2010, pp. 6, 2/1/2008, 2008.

[11] G. C. Haynes, "Dynamism Project," vol. 2010, 2009.