3-D Graphical Area Mapping Bilinear Interactive Technology [3-D GAMBIT]

BY

Ariel A. Itzhakov

A THESIS
SUBMITTED TO THE FACULTY OF

ALFRED UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

ALFRED, NEW YORK

September, 2017

3-D Graphical Area Mapping Bilinear Interactive Technology

BY

Ariel A. Itzhakov

B.S. Alfred University 2015

SIGNATURE OF AUTHOR_____

APPROVED BY_____

Dr. SEONG-JIN LEE, ADVISOR

_____

Dr. JOSEPH ROSICZKOWSKI, ADVISORY COMMITTEE

_____

Dr. WALLACE LEIGH, ADVISORY COMMITTEE

_____

CHAIR, ORAL THESIS DEFENSE

ACCEPTED BY_____

Dr. ALASTAIR N. CORMACK, INTERM DEAN
KAZUO INAMORI SCHOOL OF ENGINEERING

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Seong-Jin Lee, for the endless support of my Masters research. Without the amount of patience and motivation shown throughout the project none of this would have been possible. I would also like to thank my committee members Dr. Joseph Rosiczkowski and Dr. Wallace Leigh in their time in reviewing this thesis and assistance throughout the project. I would like to thank Rosalie DiRaimondo and Katie Decker for all their help with my random questions and for assisting me in throughout my time at Alfred. To my friends, who kept me sane for my entire college experience, thank you for all the fun times and jokes. Last, but not least, I would like to thank my parents, siblings, and the rest of my family for all the support they gave me throughout my life. Without their constant encouragement to focus on science and engineering I would not be the person I am today.

TABLE OF CONTENTS

List of Equations

Pages

LIST OF FIGURES

Pages

Abstract

The main goal for this thesis project is to develop a 3-D image scanner that will be used as part of a larger effort in robotics and obstacle detection and avoidance. The scanner developed will be added to another robotics project being done by Dr. Seong-Jin Lee, Wanrui He, and Andres Garcia at Alfred University. Known as the Bubblebot, this robot is currently being designed to expand and contract to maneuver though different size spaces. The Bubblebot will use the scanner designed and built during this thesis project to map the area and determine the optimal size it needs to be to navigate through its environment. When researching previous 3-D mapping methods, i.e. RGB-D and Simultaneous Localization and Mapping (SLAM), three main flaws in many of the current techniques were determined and corrected. First was the inability to determine the distances of the objects in the scanner's range. Second was the size of the equipment needed for the 3-D scanner. Finally, the third was the large processing power needed to run the 3-D image generator program. To correct these problems as best as possible, simple parts and programs were used including a web cam and bilinear line laser along with the Arduino and Processing programs. During the testing stage, the 3-D scanner was completed and mounted onto a sample robot for testing. With the use of a Raspberry Pi model 3B the robot was programed to map its environment, identify any obstacles in its path, and avoid them during navigation without the need of any human interaction. Overall the result of this project looked positive and when tested successfully worked. The final step for this 3-D scanner will be integrating it to the Bubblebot upon its completion.

INTRODUCTION

In this modern age of technological advancements, robots are replacing humans in many dangerous and time-consuming jobs. For example, in military situations like bomb diffusion or field reconnaissance, a robot or drone would be sent in to keep the humans safe. In the medical field, physicians use computer controlled equipment to perform surgery which requires precession. One improvement to be made in modern robots is the addition of 3-D visualization capability which would allow the robot to see the world in a human 3-D perspective. By giving robots the ability to create 3-D images of the surrounding area it could be an invaluable aid in disasters like the one that occurred to the Chilean miners during the mine collapse in 2010.

A. History of Environmental Mapping

Topographic mapping was one of the earliest forms of environmental imaging. It was created in the United States in 1884 by John Wesley Powell[1]. When first created, it was known as field mapping. Done by hand, field mapping used tape and compass traverses for distance and position, and used aneroid barometers to determine elevation and angle. All the data gathered in this method was hand drawn to make a map of the terrain with its contours. This was known as field sketching. Since these maps where hand-drawn and measured it took a significant amount of time to produce and replicate these maps. Even after the creation of the planetable and alidade, two devices used to measure vertical angles, point positions and elevation quicker than by hand the final map was still hand drawn. It wasn't until World War I did the Army Corp create the aerial photography method[2] more commonly known as photogrammetry. Photogrammetry was the use of photography in the use of surveying and mapping to measure distances between objects. This lead to the ability to make 3-D maps of the terrain by overlapping images. This method is known as stereomodel, which replaced the need to hand draw topographic maps making it easier and faster to produce and replicate. Modern day topography has not changed much since the method used in World War I, other than the addition of computers to help graph and plot the areal photogrammetry. An example of this type of map is shown in Figure 1[3]. With the addition of the computers topographic maps have become much more detailed and can be created much quicker than ever before.

Figure 1. Topographical map of Yellowstone National Park[3].

B. Literature Review

With the advancement of robotics, 3-D mapping evolved from topographical mapping to Simultaneous Localization and Mapping (SLAM) and Red Green Blue Dimension (RGB-D) environmental imaging. By using a RGB-D the camera, Haowei Liu, Matthai Philipose, and Ming-Ting Sun at the University of Washington[4], could develop a 3-D image scanner. The system used an in-depth per pixel camera to distinguish a slight difference in light and color in each pixel to create the illusion of a 3-D image. The main problem with this method is that it only generates 3-D images of items placed in front of it, as seen in Figure 2[4], and not scan an area to create a map nor could it be used to autonomously navigate a robot out of its environment.

Figure 2. RGB-D image from the University of Washington[4].

The research done by Pedro Loureco et al.[5] also utilized per pixel cameras along with ultrasonic sensors to build a quad copter that could measure distances of objects in an enclosed environment. The distance data received from the sensors were graphed, which can be seen in Figure 3.

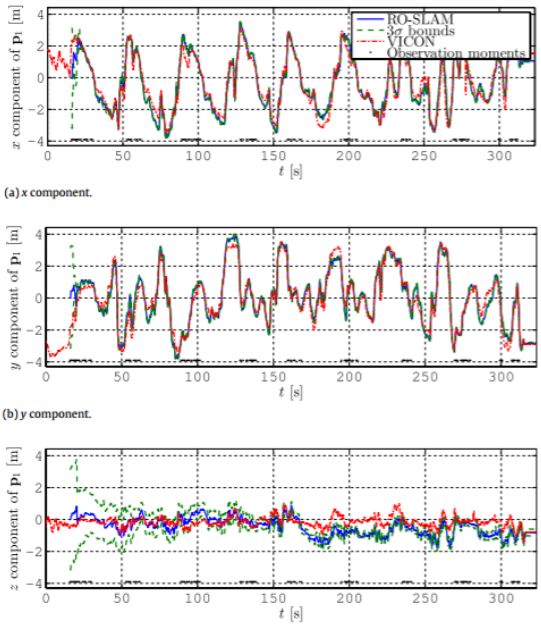

Figure 3. Sensor based obstacles[5].

This method could find the distance of the items in its area unlike many others but it too had many design flaws. For instance, it could not generate a 3-D image of the area scanned. This inevitably lead to the inability to program the quad copter to navigate through the scanned area autonomously.

Hartmut Surmann et al.[6] took another approach to 3-D mapping by using a 3-D laser range finder. This method was successful in generating a 3-D map, but as seen in Figure 4, the robot requires large equipment, like the laser range finder, making the system bulky.

Figure 4. Hartmut Surmann's obstacle avoidance robot[6].

This means larger motors were needed to run the system and a larger power supply was needed to power it.

In the research done at Tsinghua University by Xiang Gao et al.[7] a RGB-D based system was used to map out an area and avoid all obstacles successfully. The main concern with this technique was the large amount of processing and data storage needed to run the RGB-D system.

David Droeschel, Max Schwartz, and Sven Behnke of the University of Bonn[8] took both 3-D range finder and RGB-D methods and managed to create a hybrid system. This technique allowed the team to design and build a robot that could map and navigate most obstacles in its path with a small cylindrical blind spot around the robot's vertical axis. As seen in Figure 5[8], this fusion system can generate multiple map styles.

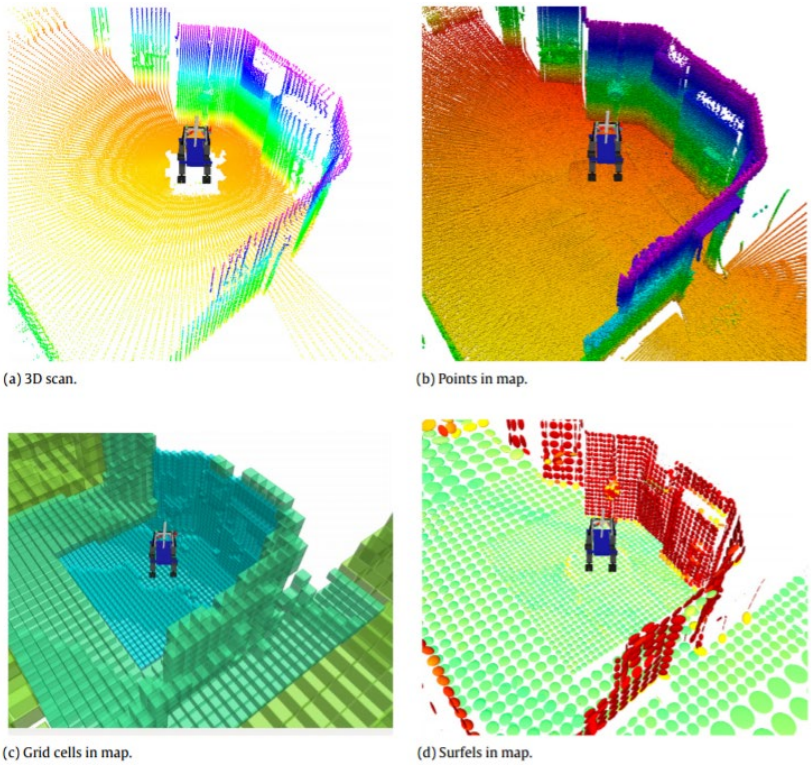

Figure 5. Multiple map styles[8].

The main issue for this technique is due to its hybrid mapping system. Since it has multiple scanners it needs a larger processing power along with a more complex computing system to allow both scanners to communicate with each other and its microprocessors.

C. History of Processing

Processing was created by two graduate students, Ben Fry and Casey Reas in 2001[9]. Its purpose was to expand the abilities basic programming into visualization. The Processing code is based off Java, JavaScript, Python, BASIC, and Lego. Many modern uses for Processing include data visualization, 3-D file exporting, and programming electronics. On a less STEM field, Processing is also used for music compositions and visual art programs in schools nationwide. The Processing software has been used by visual designers, artists, architects, and engineers alike. Work done in Processing has been featured in museums like MoMa in New York City, the Victoria, Albert museum in London, and George's Pompidou in Paris. Many exhibits use Processing to create an interactive experience with the viewers. Along with art shows it is used in many known television shows like House of Cards. The main appeal of Processing to both the art and the STEM field is the easy and free access to a simple yet essential 3-D visualization program**.**

D. Ongoing Research

The research done for this project was part of a larger robotics system designed to navigate through any environment. Known as the Bubblebot, designed by Dr. Lee and his graduate students, this robot will have omnidirectional movement, a spherical shape, and will be able to expand and contract to maneuver through different gap sizes. The ability to change its size and its unique movement will allow the Bubblebot to be prepared for any real-world scenarios that could occur. With its 3-D visualization of the area and its ability to avoid any obstacle in its path the Bubblebot could perform many tasks that are unsafe for humans to attempt. After the completion of the Bubblebot, the 3-D scanner, that was developed and built for this project, will eventually be integrated to it. A conceptual model of the Bubblebot with the scanner can be seen in Figure 6 below.

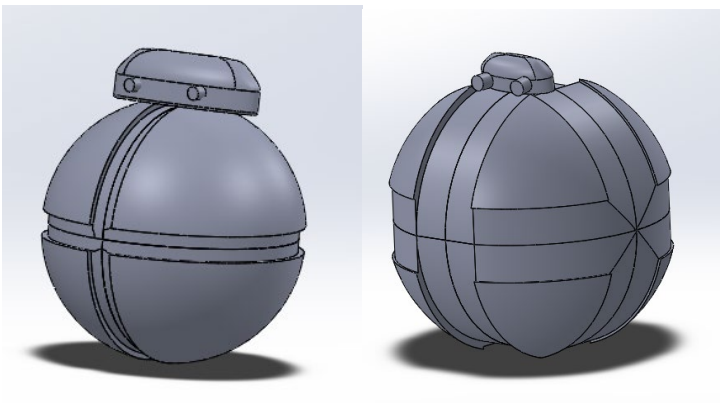

Figure 6. Conceptual model of the Bubblebot in small and large form.

The portion of the Bubblebot created for this research project was the 3-D mapping system. This consisted of designing and building a small apparatus with the capability to scan an environment, creating a 3-D image of the area, and assisting the Bubblebot to avoid obstacles using the 3-D map created. The main way this research is different from other techniques is because simpler parts are used, less motion is needed, and all coding is done with open source programs. This allows any user to replicated this process with minimal effort. In this project, many possible issues that were noted in previous research, like distance measurement and processing needs, were also corrected. To properly solve these issues and successfully build the scanner, a 3-D laser mapping approach was taken. This technique uses a simple line laser and any web cam to scan the area and create a 3-D image of the environment. Through this method the camera will be coded to only see the red light emitted from the line laser causing the illusion of a 3-D image. An example of this will be shown later. This 3-D scanning method is common in small scale item scanning and 3-D printing. When used to scan items, the 3-D Figure scanner rotate the object on a platform while the stationary camera and line laser captures every possible detail on the object. However, in the method developed for this project the laser and camera are mounted on a stepper motor. This motor will rotate the system form a 0 to 270° range of the surrounding area to find and map the items in the general vicinity. The main advantage of this method is that the parts needed to replicate the system are relatively inexpensive and the method of coding are the open source programs Arduino and Processing. Another advantage is that the power requirement to run the system is relatively low due to the minimal number of parts. One downside is that to run the Processing portion of the code to generate the 3-D image, a computer like system is need.

RESEARCH

A. Types of Mapping

The goal of this research project was to design, program, and build a low-cost image generator that creates a 3-D map of the environment and navigate a robot around any obstacle in its path. A robot with these capabilities can be used in situations which would be too dangerous or hazardous for human exploration or research. Currently there are two main techniques for this process. One being 6D SLAM using the Interactive Close Point (ICP) method and the other using the RGB – D imaging system.

1. 6D SLAM

6D SLAM mapping scans in six dimensions to create a 3-D map or image of the area. These dimensions include the positive and negative X, Y, and Z coordinates, and the roll, yaw, and pitch angles. In many 6D SLAM designs multiple cameras and scanners are used to create the map and/or image. By using 6 degrees of motion the SLAM method can scan an area from multiple angles and get a detailed image of the environment. It does this by rotating the multiple cameras and scanners in different directions and orientations and combines all the images into a 3-D map of the area. A popular use for the 6D SLAM technique includes self-driving cars. By using the multiple sensors and cameras placed around a car the computer system in the vehicle can successfully navigate around obstacles, like other cars and pedestrians[11]. Similar research was done by Nüchter et al. at the University of Osnabrück[6]. In their system, multiple scanners were used to create a 3-D map of an area. One issue with this technique is the way the distances to the obstacles are measured. The robot will measure the distance to an obstacle by driving to it at a set speed and calculating the distance from its starting point based on the time it took to reach it. After this, the robot will use the new information to calculate the initial estimations of the distances to other objects based on its original starting point. However, the robot will still drive up to other obstacles in its original map and find the precise distances from the starting point. This is done to adjust the formula, shown in Figure 7[6], used to calculate the objects in its surrounding area.

$$\mathbf{R} = \begin{pmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y + q_z q_0) & 2(q_x q_z + q_y q_0) \\ 2(q_x q_y + q_z q_0) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_x q_0) \\ 2(q_z q_x - q_y q_0) & 2(q_z q_y + q_x q_0) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{pmatrix}$$

Figure 7. Formula used to estimate distance[6].

This method of 6D SLAM creates a loop of the area to be able to map the system and get an accurate distance measurement of all obstacles. Multiple scans of the area are made and overlapped to create a map of the area before the robot can navigate the environment and calculate distances. This technique is called pairwise matching which allows the system to get a more accurate readout of the map shown below[6]. As seen in Figure 8, the area scanned using the pairwise technique becomes more detailed with every scan.

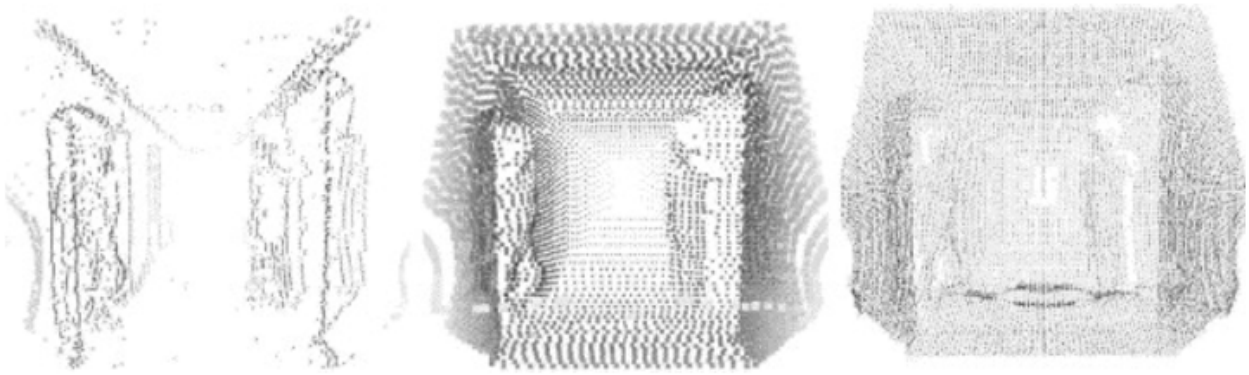

Figure 8. Maps using pairwise matching[6].

Through pairwise matching the system could only generate a map of one area at a time. Due to the need for multiple scans, calculations, and measurements of distances from obstacles from the starting point a longer time is needed to an accurate 3-D map of the environment. Another issue in this method of 3-D mapping and imaging is the need for multiple scanners and sensors. The use of more parts increases the likelihood of equipment malfunction. The increased amount of parts will also affect the robot's size and the power needed to run the system. The main advantage of pairwise matching is even though the system has multiple scanners and sensors very little processing power is needed to run the program and generate a 3-D. In most cases like this one, only a microcontroller is needed to run all the hardware and software needed.

2. RGB-D

With today's technology, the simplest and quickest technique for 3-D imaging and mapping would be the RGB – D method. This is because Microsoft created the Kinect to use for their video game system. The Kinect is a small device that creates an in-depth per pixel image of the surrounding area using an IR light camera and an in-depth pixel camera as seen in Figure 9[12]. The main reason for its use and popularity is due to its low cost and the accessibility of the device. The RGB – D method of 3-D mapping and imaging uses color images with per pixel depth. By distinguishing the color of each pixel in the image, the RGB – D system can create an in-depth 3-D image or map of the surrounding area.

Figure 9. Microsoft Kinect[12].

The research done by Xiang Gao and Tao Zhang in the Department of Automation, at Tsinghua University, in Beijing, China developed a successful method in using RGB-D equipped robot to map and navigate an enclosed area[7]. Since the RGB – D technique uses an in-depth camera to create the 3-D image of the area this system will need a preset database of the material textures to properly create a 3-D image of all objects in the mapped the vicinity. The main problem found using the RGB – D method was since the system needed to determine the texture of the obstacles, the time needed to map the area would be much greater than other methods. The current solution used to avoid this problem is by decreasing the resolution of the image created. Doing this creates problems of its own. A significant decrease in resolution decreases the

accuracy of the 3-D map as well. This will also limit the distance the in-depth camera can map at each run. Even with all this, the main concern that the RGB – D mapping system has is that, even with low resolution imaging, the processing power needed for this system is much larger than any other method. For the RGB-D mapping and imaging process to work, a laptop is used as the main source of computing for the system as seen in Figure 10[7]. Due to this, the robot using this system will need to be much larger to accommodate the laptop and to compensate for the increase in weight. The power supply for the system will also be much larger due to the larger robot and computer system.

Figure 10. Turtlebot[7].

Final problem with the method of RGB – D mapping would be that there is no system or procedure to measure the distance to the obstacles in the mapped area.

By researching and learning from these techniques used in previous studies created the opportunity to design a new and simpler method of 3-D imaging. The first step taken for this was to design the system and select the proper hardware needed.

B. Hardware

After comparing the previous research three main flaws were determined. First was the inability to determine the distances of the objects in the scanners range. Second was the size of the equipment needed tor the 3-D scanner. Finally, the third being the large processing power needed to run the 3-D image generator programs. The method that was designed for this project was a simple laser scanner along with an ultrasonic sensor. The laser scanner will consist of a single line laser and a web cam which will be programmed to only record the red light of the laser. The ultrasonic sensor will be used to measure the distances of the objects in the scanned area. Through this the first two problems in many modern scanners could be solved. To solve the large processing requirements a new code was written using the program Processing. Since the code running the Processing program requires low computing power, the program could be placed on a simple microcomputer like a Raspberry Pi.

The objective in this section was to design and select the parts to create the 3-D mapping equipment needed for the system. These parts would have to be lightweight, energy efficient, low-cost, and compact.

1. Microcontrollers and processors

To run the stepper motor, motor driver, camera, line laser, and ultrasonic sensor used for this project and create the 3-D map the system needed to be controlled and told how and when to operate. The simplest and most cost-efficient way to do this was with an Arduino microcontroller as seen in Figure 11. An Arduino is one of the most popular microcontrollers available to programmers that uses open source coding to allow users to create any code needed or build off a

pre-existing code. Each Arduino was powered by a 7.5-volt power supply consisting of five AA batteries.

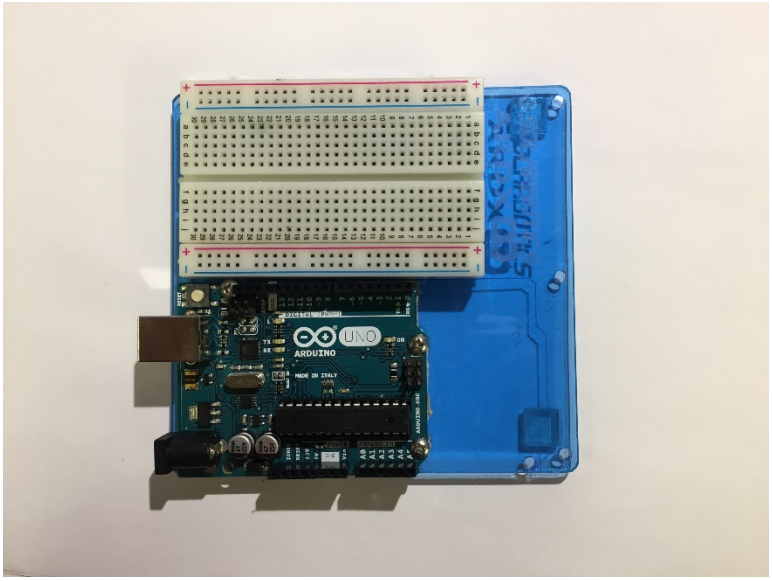

Figure 11. Arduino microcontroller.

In addition to the Arduino to run the hardware an additional microcontroller was needed for processing all data obtained from the scanner. To run the 3-D mapping system a Raspberry Pi model 3B was selected. A Raspberry Pi in its most generic definition is a microcomputer. The entire thing consists of one chip that contains a processor, USB slots, a micro SD slot, a micro HDMI power input, and an HDMI as shown in Figure 12. This microcomputer was selected due to its popularity. This helped in resolving any bugs that occurred during programming. Since the Raspberry Pi also runs on open source coding no additional programs were needed to be purchased to run the system. Since the codes used to scan and generate a 3-D map were developed on Windows based programs the microcomputer that would be used would also need to run Windows and the Raspberry Pi offered Windows 10 as one of its operating system options. The Raspberry Pi model 3B was used to process the data obtained by the 3-D scanner.

Figure 12. Raspberry Pi model 3B.

The most challenging issue from using the Raspberry Pi 3B was its power requirements. Like an Arduino, the Raspberry Pi 3B runs on a 5-volt 1 amp, but unlike the Arduino the Raspberry Pi 3B power input, is a micro HDMI slot. For the first attempt to power the Raspberry Pi 3B a micro HDMI wire was spliced and attached to a battery pack similar to the one powering the Arduino. This worked for a short time. However, it was discovered later that, even though the power input was equivalent, the Raspberry Pi 3B would need a significantly larger power supply. This was because processing the data from the scanner took more power than first estimated. To resolve this problem there were two main options. One was to increase the size of the battery pack, but this would increase the weight and overall size of the scanner. In addition, the constant need to replace the batteries would create an unnecessary and easily avoidable cost. The second option was to get a portable phone charger to run the Raspberry Pi 3B. The main disadvantage in using a portable phone charger was its initial cost and need for recharge time. Where a battery pack can be instantly changed, a portable phone charger would need at least an hour for a recharge time. Overall, it was concluded that a smaller, light weight, and longer lasting portable charger would be more efficient than a battery pack.

2. Stepper motor and controller

One of the key parts of designing and selecting the hardware for the system was the selection of the motor that would be used. There were many options to choose from, for example basic two wire motors, three wire servomotors, or stepper motors. There were a few requirements that need to be met to be able to find the right type of motor. The main requirements were that the motor needed to be easily programmable, have enough torque to move the system, have low power consumption, and be structurally strong enough to hold all parts place on top of it. A two-wire motor could not be used since it is more difficult to program. Many motors like servo motors have built in motor controllers but since regular two wire motors have little to no circuitry there is no way to regulate the power that goes into it. This makes controlling the speed and direction of the motor spin more complicated than any other type of motor. A servomotor like mentioned before has the motor controller built in. This allows the user to preprogram its speed and direction. This meant that the servomotor did meet all the requirements but since the amount of weight that would be placed on top of the motor and the amount of torque needed to move it was still unknown it was difficult to identify an appropriate servo motor. This was since servomotors unlike to wire motors and stepper motors use built in gearboxes to increase speed or torque. It would be difficult to select the correct gearbox assembly before the system was complete. A stepper motor also met all the requirements needed and due to its ability to vary its torque and speed outputs given its wider voltage range it had a better chance of accommodating the unknown load. Even though the stepper motor did not have a motor controller built in its ability to vary its speed and torque outweighed this problem. The stepper motor selected was the NEMA 16 motor as seen in Figure 13.

Figure 13. NEMA 16 stepper motor.

The one complication with the stepper motor was selecting a proper controller and power supply for it. There are many options when choosing the proper controller for this stepper motor. These options would very in cost from anywhere between $2 and $150 and could be programmed using many different platforms. For the system, a A4988 driver was used since it was the simplest one to program with Arduino and most cost-efficient at a price of only $3. As seen in Figure 14 the A4988 driver is a small motor controller with 16 pins. For this project, only 12 pins will be used. Two for power, four for the stepper motor, four to connect to the Arduino, and two to reset the system after use. This will be discussed further during the equipment set up.

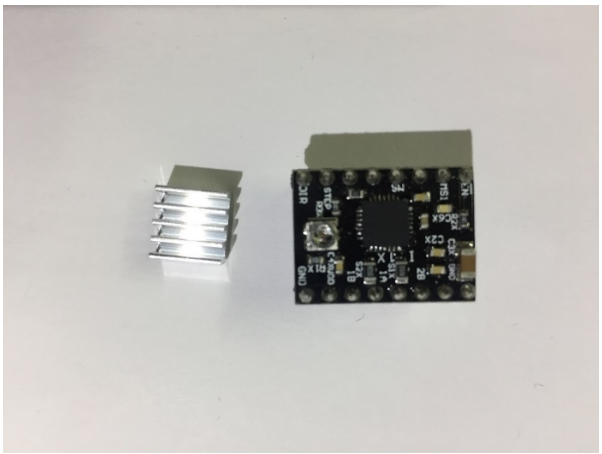

Figure 14. The A4988 driver used to power and control the stepper motor.

This controller takes a separate power supply to run the stepper motor. This was needed because the stepper motor operates anywhere in between 5 to 18 volts, and Arduinos are only able to safely give off 5 volts. For the system two 9-volt batteries were connected to run the stepper motor.

3. Scanner

Building the scanner was quite simple. There are three main parts for the physical scanner. One being any run-of-the-mill webcam shown in Figure 15. The webcam was used to record the red light from the line laser and block out the rest of the area. A lower resolution webcam was preferable since it would detect less ambient red light in the mapped area. For this project, a Logitech C270 webcam was selected since it was the easiest one to acquire at the time.

Figure 15. Logitech C270 webcam.

The webcam selected had a USB output cable to connect to computer so it could send and receive data and receive power. This means that the webcam needs a 5-volt 1-amp power supply. For initial testing, the webcam was powered by the USB port on the computer running the 3-D mapping program and then later through the Raspberry Pi USB input.

4. Line Laser

The next part to the scanner is a line laser shown in Figure 16. This is like any ordinary laser pointer that shines a red light just like any other, laser but instead of a point, it produces a vertical line. The need for a line laser was to allow the camera to scan one line in the area at a time which helped create a more detailed map.

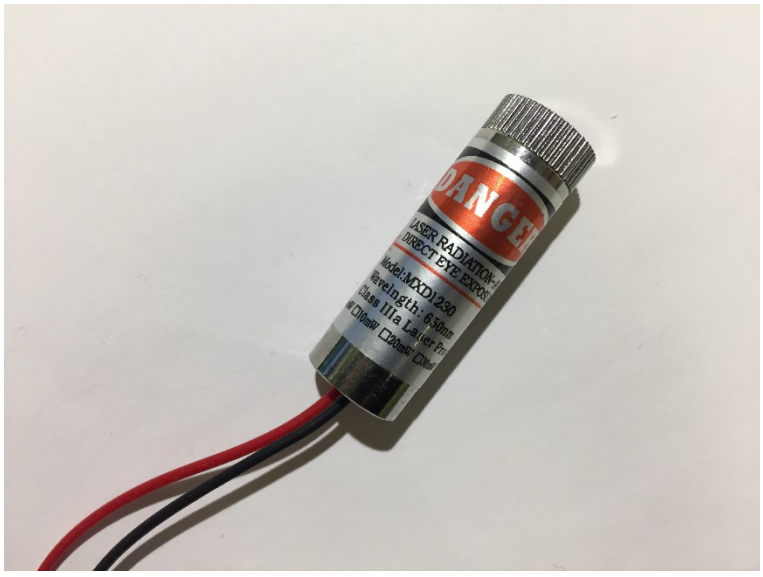

Figure 16. Line laser used for the 3-D scanner.

This laser needed 6 V to work and was also given an independent power source consisting of four AA batteries. The main reason for its own battery supply was to prevent risk of the laser drawing too much power from the primary power supply connected to the Arduinos and Raspberry Pi.

5. Ultrasonic Sensor

Finally to tell the distances of the obstacles in the 3-D map an ultrasonic sensor was used. For the set up the HC-SR04 ultrasonic sensor shown in Figure 17 was used.

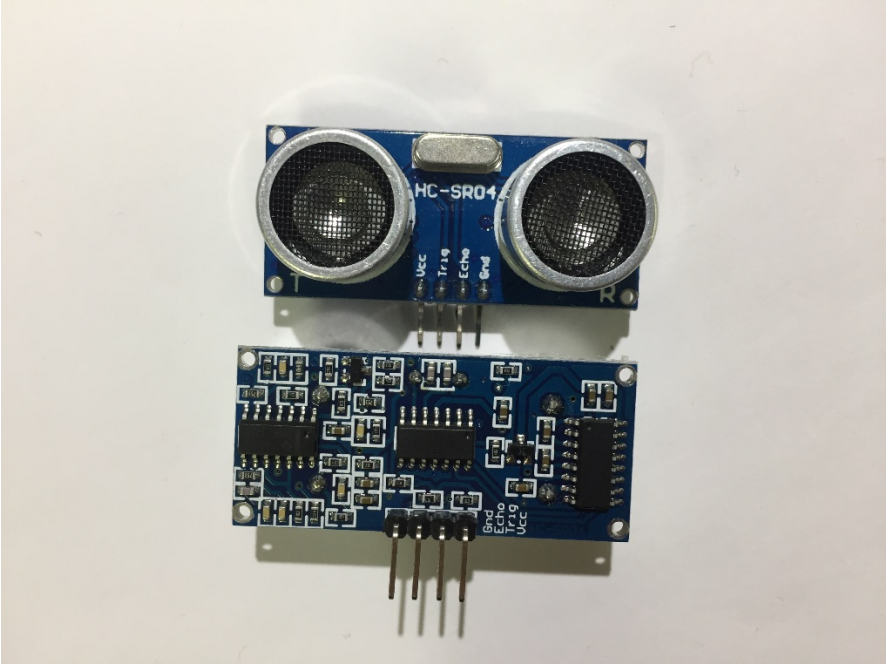

Figure 17. HC-SR04 ultrasonic sensor.

This ultrasonic sensor was selected in particular because of its small size and ability to detect objects up to 6 meters away. Also, the HC-SR04 is the most popular when using Arduinos to program. Finally, this ultrasonic sensor is sold for only $3 where others that have similar performance vary for over $200. This sensor has four pins, two for power and two for data transfer to and from the microcontroller to which it is connected. The HC-SR04 ultrasonic sensor takes a negligible amount of power so it was connected directly to the microcontroller used in the system.

6. Assembly

To assemble the 3-D scanner for testing the first thing part acquired was the wooden board that all the circuitry would rest on. This board was approximately 40 cm in length and 5 cm in width. These dimensions were selected to hold all parts like the camera, line laser, and Arduino. The web cam was placed at the first 4 cm of the left side of the board at a 30 to 40° angle. On the right side of the board approximately 1 cm from the edge place and secure the line laser. In the

center of the board which in this case would be 20 cm from either end drill a hole slightly smaller than the shaft of the stepper motor being used. The board and the stepper motor need to fit tightly to prevent the system from slipping while scanning the area. Place an L bracket approximately 2" x 2" in size on the left Side of the board with the vertical side facing the stepper motor shaft. This will be used along with a belt to stabilize the system and prevent any unnecessary shaking. Entire system can be seen in Figure 18. When added to the final robot being designed by the rest of Dr. Lee's team all the hardware will be concealed in an organized fashion.

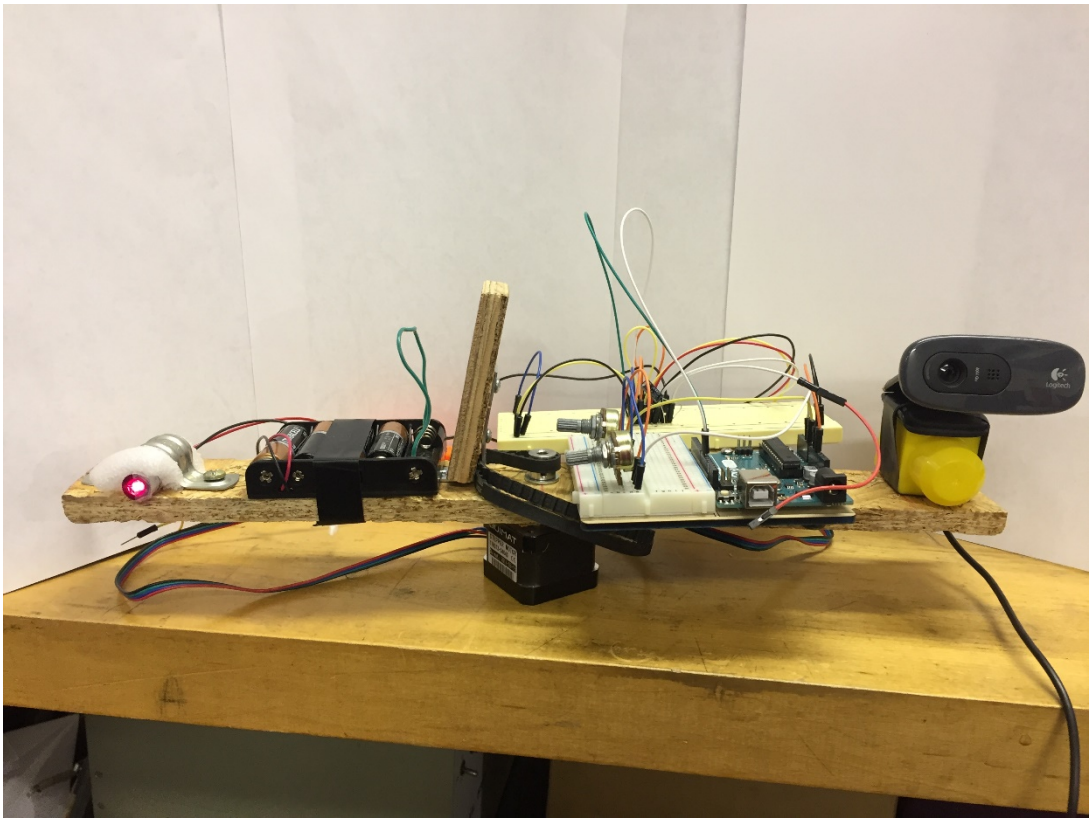

Figure 17. Scanner setup.

7. Stepper Motor Controls

The first step in the electrical assembly of the system was to wire stepper motor and its power source to the motor controller. After that the motor controller was wired to the Arduino along

with two variable resistors, which control the speed the stepper motor removed. This all can be seen in Figure 19.

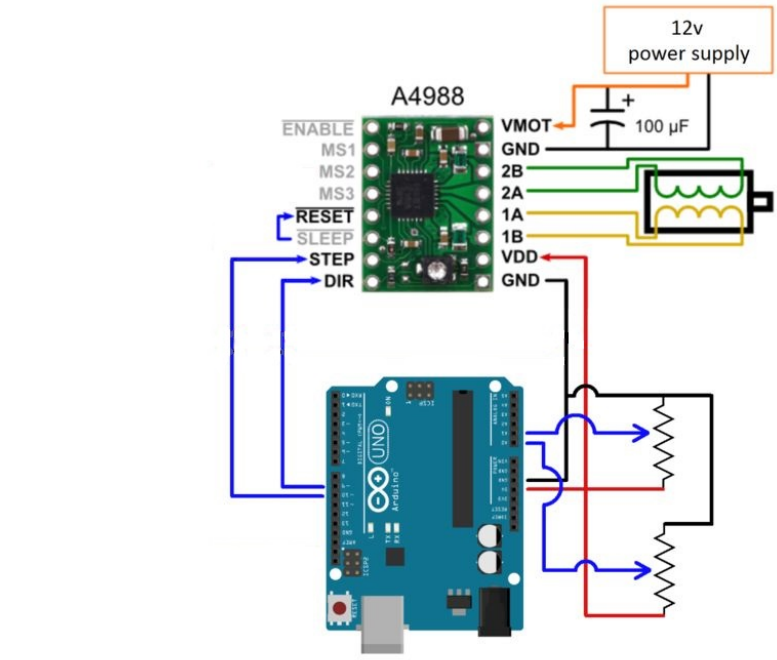

Figure 19. Arduino stepper motor and motor controller setup.

Figure 20 shows the actual Arduino setup.

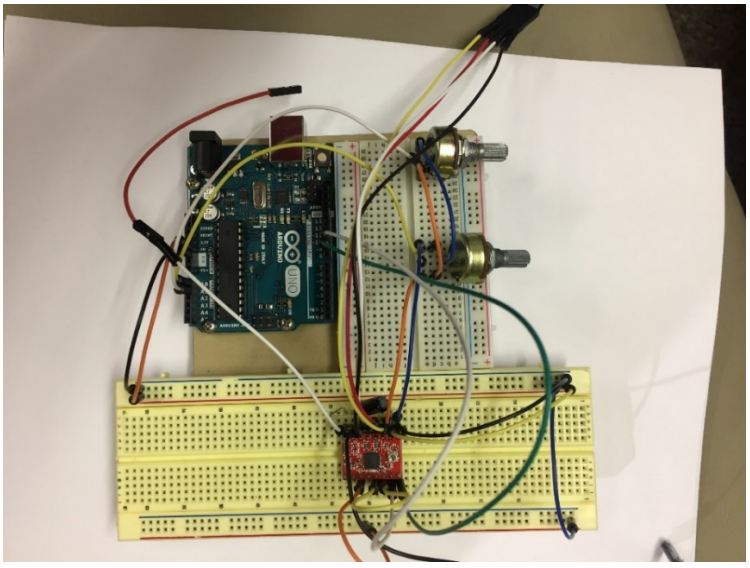

Figure 20. Electrical assembly.

8. Budget

The estimated budget for this project was approximately $252. This includes all the parts mentioned, any other hardware needed for the robot, and an additional $50 for miscellaneous expenses. The complete breakdown of the budget and all expenses can be found in Appendix G.

DESIGN

A. Ultrasonic Mapping

When starting this project, the first approach taken to understand 3-D imaging using the HC-SR04 ultrasonic sensor. The first step taken to understand how the ultrasonic sensor worked was to create a code using Arduino that would use the sensor to measure distances of any object it was pointed towards and display the information obtained. The HC-SR04 ultrasonic sensor has four pins. The first and last pin are used for voltage and ground both obtained from the Arduino. The other two pins are trigger and echo. The trigger pin is what sends out the ultrasonic sound wave and the echo pin is what receives the sound wave. It can determine the distance the object in front of it is by taking the time sound wave traveled from the ultrasonic sensor and bounce back to it multiply it by the speed of the soundwave and divided by two. All this could be seen in equation 1.

$$cm = \frac{duration * 0.034}{2}\ , inches = \frac{duration * 0.0135}{2} \quad (1)$$

Equation 1. Ultrasonic distance formula in metric and imperial.

The speed that the sound wave travels may differ for each ultrasonic sensor model but for the one used in this project it was 0.034cm/s. If imperial units are required the speed of the sound wave in inches would be 0.0135in/s. The duration in the formula is the time it takes for the soundwave to reach the object and return to the echo pin. The reason the initial distance is divided by two is because the time being used is from the ultrasonic sensor to the obstacle and back which is double just to the object being measured. The next step was to display distances measured in an easily readable and accessible area. Since the coding was being done on Arduino the simplest solution for this would be using the Serial Monitor function. This allows any information received from sensors received by the Arduino to display only when requested. The coding needed for this could be seen in Figure 21.

```
void setup() {
  //Serial Port begin
  Serial.begin (9600);
```

```
  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();
```

Figure 21. Arduino serial print code.

The result for the Serial Monitor showed the distance of the objects the ultrasonic sensor was measuring in both inches and centimeters shown in Figure 22.

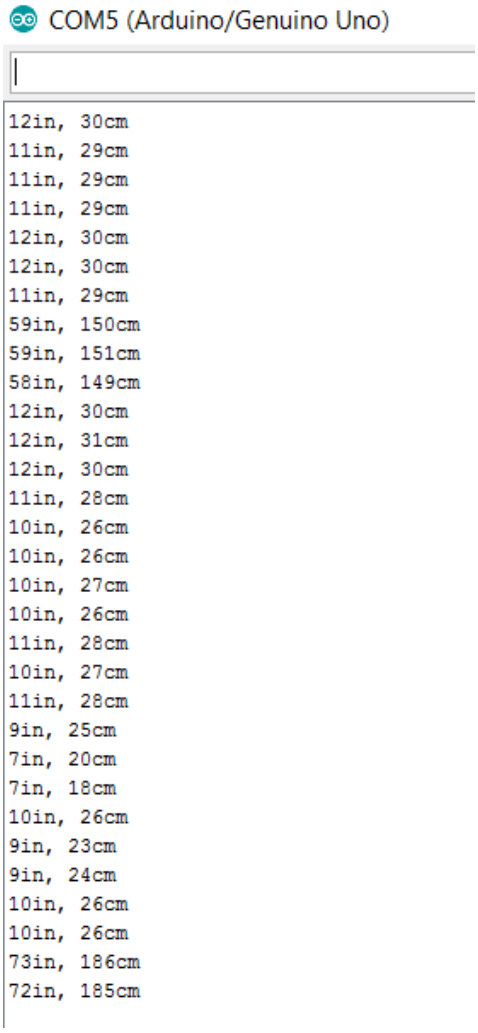

Figure 22. Arduino serial monitor.

The next step taken with the ultrasonic sensors was to see if it would be possible to scan an area and see if it could create a 2D map. The first issue that arose when attempting ultrasonic 2D mapping was the inability of the Arduino software to create an image as an output. To solve this, another program would be needed that could communicate with the Arduino software and hardware. The software selected to solve this problem was Processing version 3.1.1. The reason an older version of the Processing program was selected was since it had the larger libraries for video and image drivers and generators. Even though another program was being used the Arduino software was still needed. The Arduino software was used to control the Arduino and all other hardware connected to it while the Processing software would take all data and generate a two-dimensional Polar coordinate (R, θ) map of the area. The Arduino software and the Processing software would communicate through the com port and any information Processing needed was obtained through the Serial Monitor. Figure 23 shows the code needed in Processing to allow both software is to communicate to each other.

```
myPort = new Serial(this,"COM4", 9600); // starts the serial communication
```

Figure 23. Arduino to Processing communication code.

To be able to get a full scan of an area the ultrasonic sensor needed to be mounted on to a servo. Along with rotating the sensor, the servo was used to measure the angle at which the system was oriented. The servo and sensor setup can be seen in Figure 24.

Figure 24. Ultrasonic sensor mounted on a servo.

This information was recorded in the Serial Monitor and relayed to Processing as well. The final 2D scan showed all objects and the distance from the ultrasonic sensor. Since the ultrasonic sensor scanner was set up to only scan a certain height, the final map had a variable R coordinate and θ coordinate. An example of the 2D map can be seen in Figure 25.

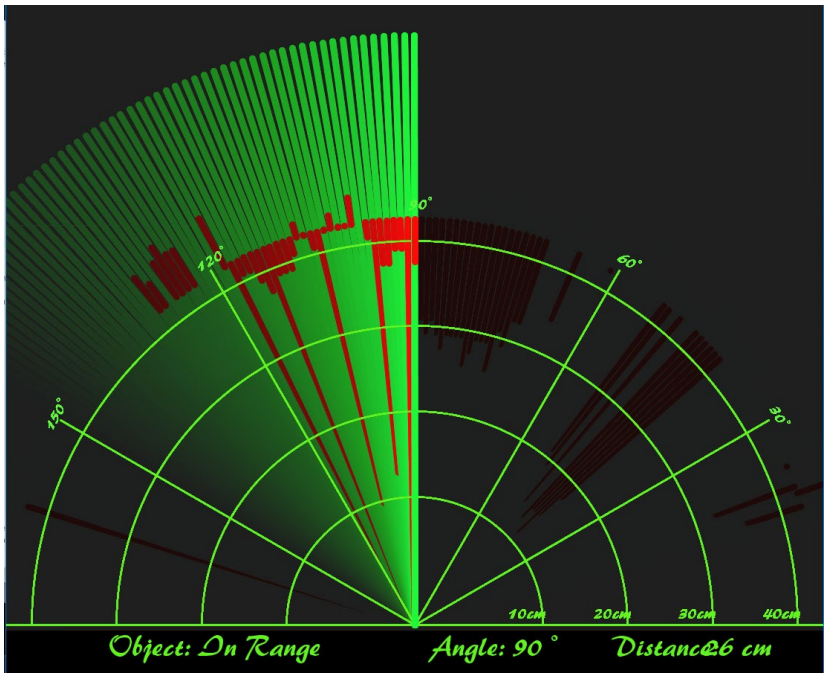

Figure 25. Processing 2D range finder.

The next step taken with ultrasonic sensors was to attempt to create a 3-D map using similar techniques as the 2D method. The approach taken was by adding a second servo to the original system and by adjusting the Processing and Arduino code it was determined that using ultrasonic sensors alone to create a 3-D map would not work. Any image generated through the ultrasonic sensors came out to blurry and boxy to understand. When using the ultrasonic sensor mapping approach, it would only work when the sensor would scan small objects at a time. Unless zoomed in on each small feature nothing in the image could be made out.

B. Camera

In the next attempt to create a system that could generate a 3-D image of the video camera approach was used instead of ultrasonic sensors. In other research like SLAM and RGB-D multiple cameras and sensors were used but for this attempt only one simple web cam was used. The final goal with this approach was to use a web cam and a red line laser to scan the room and create a 3-D image of the area. The way this would work is that through Processing the web cam would ignore all light except for the red light given off by the line laser. By ignoring all other

light in the area, the camera would not show or see anything in the area being scanned other than the laser. An image of this can be seen in Figures 26a and 26b.

Figure 26a. Area scanned.

Figure 26a is an example of what the red line laser and camera sees before the code used to generate the map is used. Figure 26b, on the other hand, is what the camera sees after the 3-D image generator code on Processing is activated. When both are compared before the program is running all light is being seen by the web cam but after only the red light is recorded.

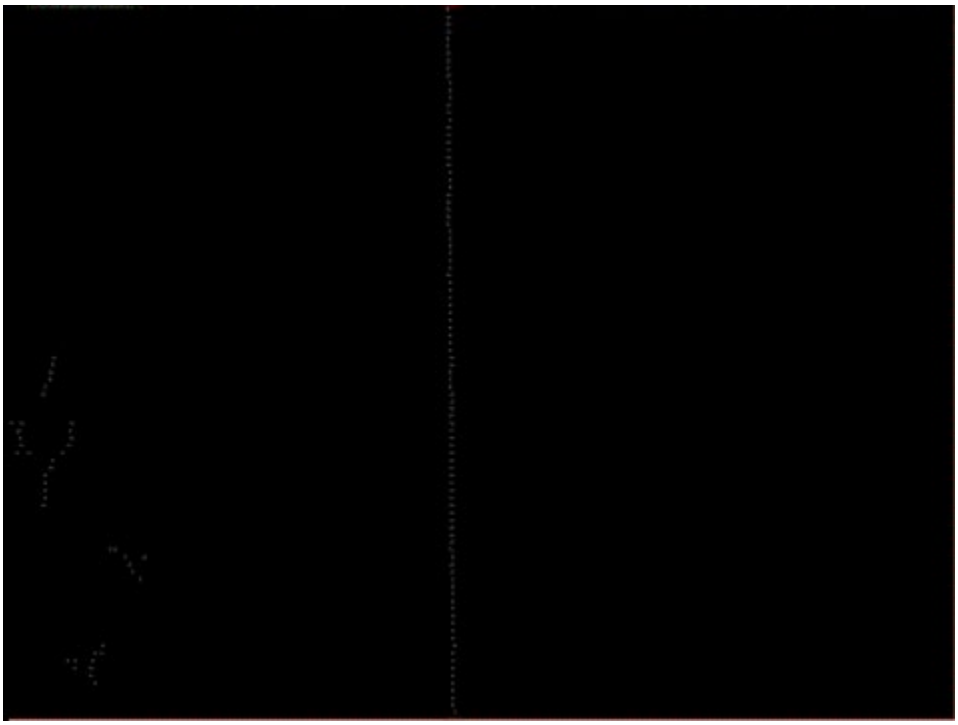

Figure 26b. Area seen.

To completely scan the environment and for Processing to know where the object scanned, the system would have to rotate and inform the Processing software at which angle the red line laser was on. This was done through Arduino. The angle of the motor used was monitored through the Serial Monitor and sent to the Processing code through the com port like before. Processing finally took all the data from the camera and the Arduino to create a text file. The information obtained in the scan was split up by commas and semicolons. The commas would represent different pixels in one line scanned at its current angle, while the semicolon would signify the starting of a new angle and therefor new line of data points and pixels. The first number after a semicolon would be the current angle the scanner was currently set to. The total time it took for a 0 to 270-degree scan to be completed is approximately 20 seconds. A small portion of this file is shown in Figure 27. On average, the amount of data points and numbers in the text file is usually well over 800,000 and is so large it would take up over 200 pages of a standard word file.

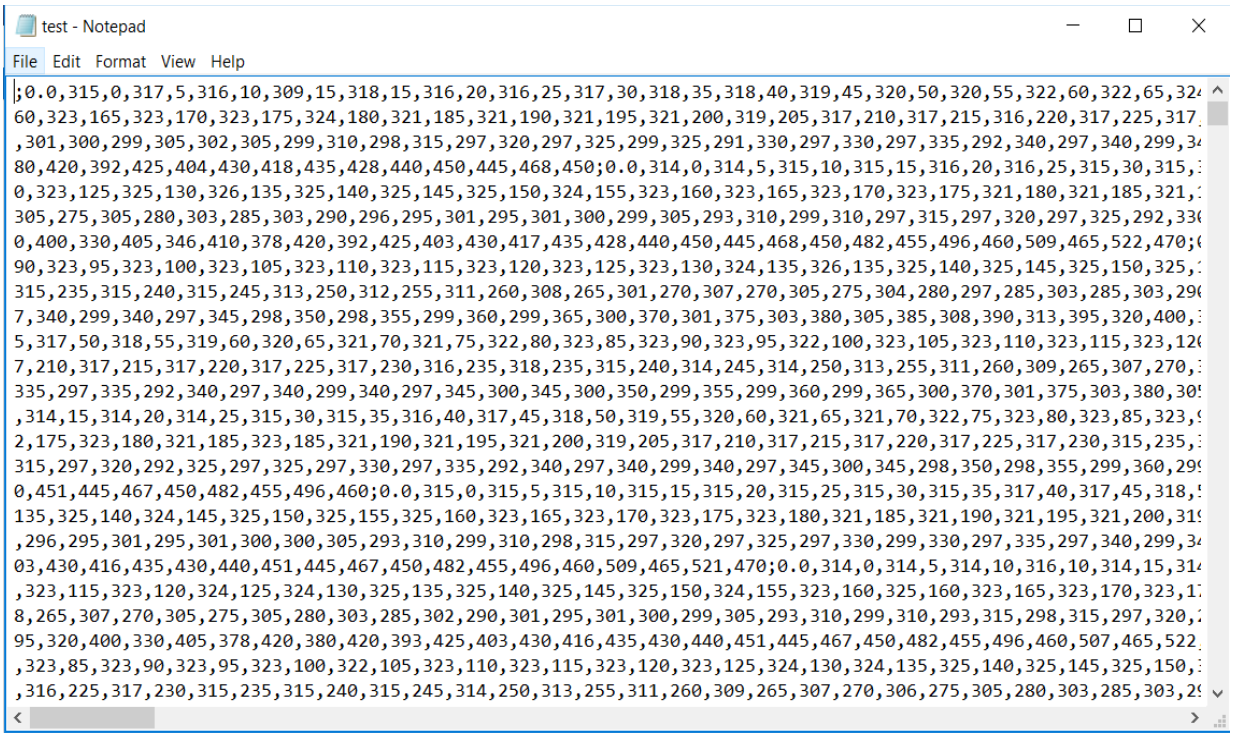

Figure 27. Text file generated by scanner.

When the system completes its scan, the motor used to rotate the camera and line laser would reset the scanner to its initial position so it would be ready to run when needed. Even though the camera was only set to view the red light and to ignore all other information, the number of pixels and data points that were received was overly excessive. The large amount of data would later cause difficulty in generating an image of the area. In most scenarios, there was so much data for the Processing program to run through that it would cause the software to freeze or crash all together. This problem was solved by decreasing the number of pixels the camera would use. This could be seen in Figure 28.

```
if (k > 0) {                    // check if "k" found any white pixels AND ignore a specific dead pixel
  img2.pixels[i-(k/2)] = color(255);  // draw a white pixel in image 2 at the middle of a row of white pixels from image 1
```

Figure 28. Decreasing captured pixels.

The way to decrease the number of pixels used by the camera would be by dividing k, the number of pixels. For the camera that was used for this project only half the pixels were used to

avoid data overload. Even though the number of pixels are being decreased to decrease the overall processing power, the final image would still be clear enough to make out. the overall number of pixels an average camera has is quite large but would vary from camera to camera. The one used for this setup has anywhere from 1.2 to 3.0 megapixels depending on the size of the image being taken.

C. Stepper Motor

To rotate the hardware in the system to accurately scan the area a motor was needed. The motor angle would need to be monitored as well so the data points scanned by the laser and camera, and be arranged in the correct location when creating the 3-D map. Both tasks were achieved through the Arduino. A stepper motor was used in this situation along with a A4988 motor controller. The controller regulated the power input to the motor, which direction the motor was spinning, monitored the stepper motor's angle, and displayed the angle in the Serial Monitor. To set the desired max angle, the stepper motor would spin the Arduino would have to be coded accordingly. This can be seen in Figure 29.

```
void loop(){
  del = map(analogRead(pot1), 0, 1023, 500, 50);  // reading the pot1 value and mapping it to the delay
  spd = map(analogRead(pot1), 0, 1023, 1, 100); // reading the pot1 value and mapping it to the speed
  angleEnd = map(analogRead(pot2), 0, 1023, 0, 180);  //// reading the pot2 value and mapping it to the furthest angle
```

Figure 29. Arduino angle adjustment.

In most scenarios tested the angle was not needed to be larger than 180 degrees, but if needed, the scanner can be set to rotate a full 360 degrees to map a room or enclosed environment. The number in these lines of code that would be changed if the angle needed to be adjusted would be the fourth number in the last line of code.

D. Image generator

As seen from the text file the information obtained from the 3-D scanner was too complex to decipher manually. To fix this problem another Processing code was made to decipher and generate the desired map of the area. This code generates an X, Y, Z grid where all data points

obtained from the original Processing code will be placed. A blank version of the grid is shown in Figure 30.

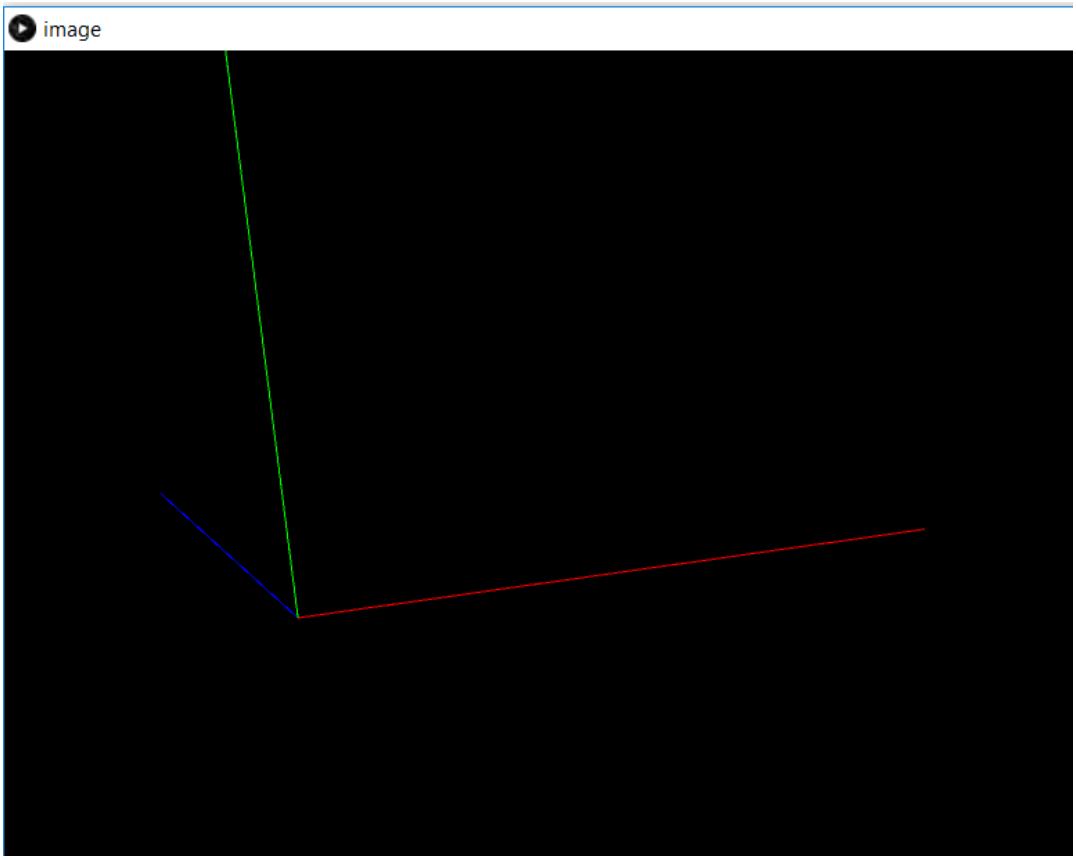

Figure 30. Blank generated image.

When making the imaging generator code, the main issue that occurred was a system overload. The initial map generating program would attempt to automatically generate the 3-D image all at once when activated. This would cause Processing, and on occasion the computer, to freeze or crash due to software and hardware limitations. The way this issue was resolved was by adding an "if" function that caused a manual activation of the 3-D image.

By adding this function into the code, it allowed Processing time to analyze all the data from the scanner and correctly rearrange it without overloading or crashing.

When viewing the image generated in some situations the orientation of the image would need to be changed. For instance, if examining a 360-degree image of an enclosed area the orientation

should be moved up the Y axis. To do this the numbers in the code, shown in Figure 31, should be adjusted accordingly.

```
camera(50*cos(mouseX/100.0)+10, 20, -50*sin(mouseX/100.0)-30,  // X,Y,Z location of the camera (not the webcam)
      10, 0, -30,                                              // X,Y,Z location of what the camera is looking at (not the webcam)
      0.0, -1.0, 0.0);                                         // where is up for the camera (not the webcam)
```

Figure 31. Image orientation code.

To move the imager perspective in the X axis the first number in the second code would be changed. For the Y axis, the second number in the third line of code. Finally, for the Z axis the third number in the second line would be adjusted.

TESTING

A. Results

When testing the laser scanner, the main issue that arose was being able to control the speed at which the stepper motor was moving. If too slow, the scanner would take too long in time out causing it to create an incomplete 3-D image of the area. If too fast, the scanner would run the risk of skipping important data points or marking pixels in the wrong angles and positions. To solve this problem two variable resistors, each approximately 500K, were added to the system. One of the variable resistor's will control the max angle the scanner would rotate and the other would control the speed at which the scanner would run. Doing this allowed for manual adjustments to the system without having to change the code every time the scanner would need to be adjusted. The speed and angle at which the stepper motor would run is needed to be set before the scan is initiated. In most situations, the scanner was set to rotate at 180° and to spin at the maximum speed that it's power supply could provide. An accurate scan of a small area can be seen in Figure 32.

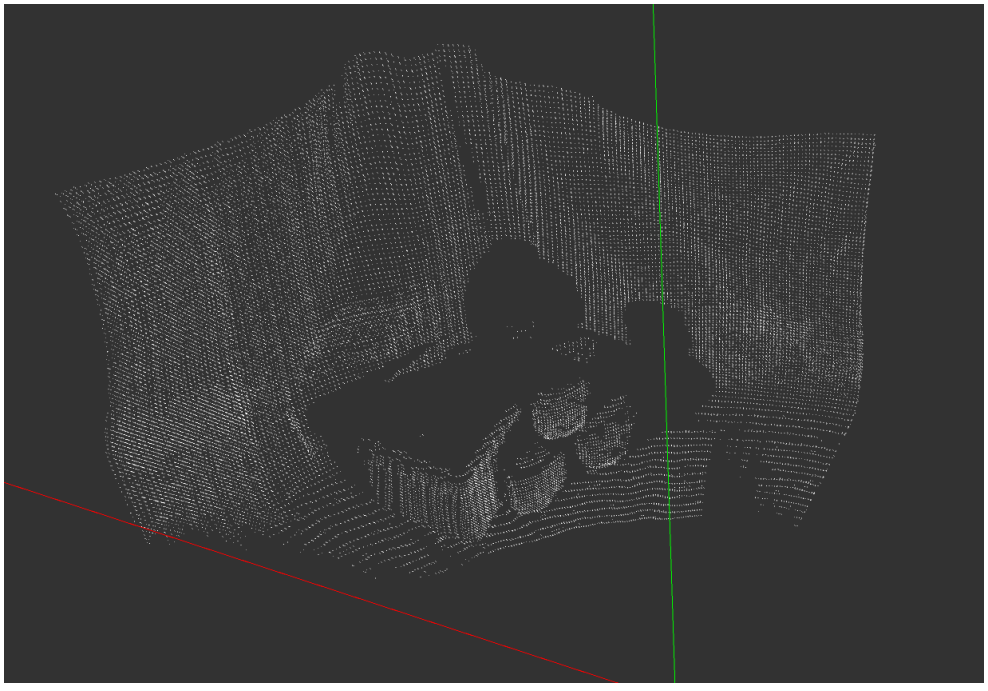

Figure 32. Image generated.

In the scan, there are a few everyday objects that were found and arranged to test the scanners accuracy. The main shapes of these objects were basic square and circular Figures. These basic shapes were selected to mimic most real-world objects that tend to be either rectangular, spherical, or cylindrical. An image of the items scanned can be seen in Figure 33.

Figure 33. Image scanned.

As seen in the above two images the scanner could mimic the depth of the objects by seeing the light intensity of the laser. The closer the red light was to the camera the brighter it was. This would help create the illusion of a third dimension. Even though the scanner could make a functional 3-D image of the objects scanned it still couldn't know the distances of the objects from its initial position. To solve this the ultrasonic sensors used in the beginning of the project would be combined with the laser scanning system to both create a map and know the distance of the items in the area.

B. Using the Scanner with a Robot

After the scanner was tested and in working order the final step was to place the system on any land base robot and see if it could be programmed to scan and avoid the obstacles in an area. The robot used for this simulation was a simple and small car as seen in Figure 34. This was done only to test the scanner and to make sure it will work later on for the next step in Dr. Lee's research.

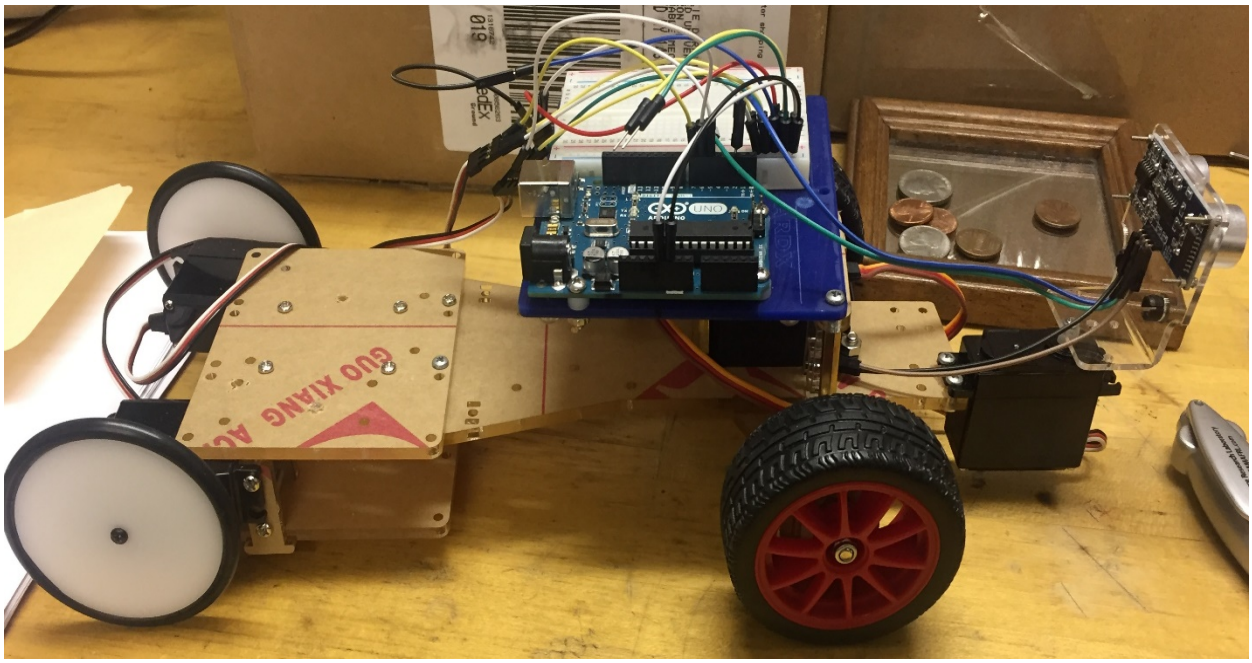

Figure 34. Robot used.

The first issue that arose when adding the scanner to the robot was to design a method in which the Arduino and the scanner could be both attached to the robot. Since the scanner would need to see its surrounding area it would have to be mounted above all parts of the robot to see past any extruding items on it. A model of the scanner's stand can be seen in Figure 34. The stand was designed specifically for the dimensions of the stepper motor being used on the amount of space on the robot. This would need to be adjusted for any other robot or stepper motor.

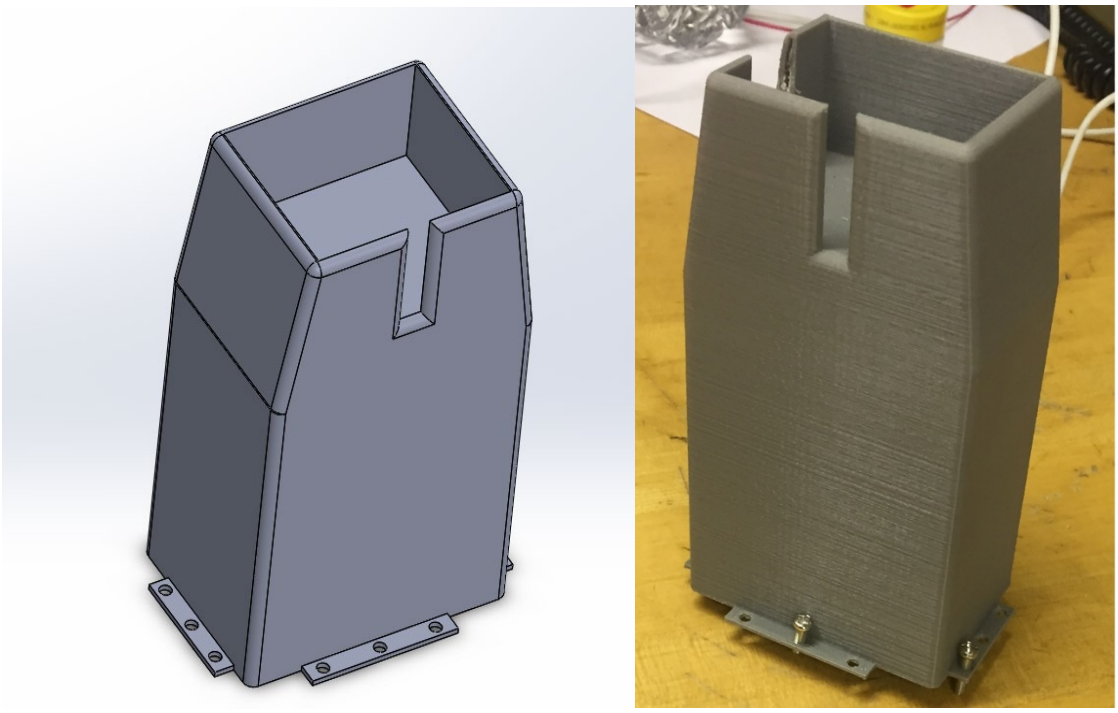

Figure 35. Stepper motor stand.

The next issue that arose was to control the robot's motors to move forward and avoid obstacles. To do this a secondary Arduino was added to the system. This Arduino would only control the two servo motors controlling the tires and the one servomotor controlling the ultrasonic sensor. The main Arduino on the scanner would communicate with the secondary Arduino using the last two analog pins on each Arduino. This set up is known as a master slave configuration since the main Arduino, in this case the scanner, would control the output of the secondary Arduino, in this case the servo motor controller. A diagram of this set up can be seen in Figure 36.

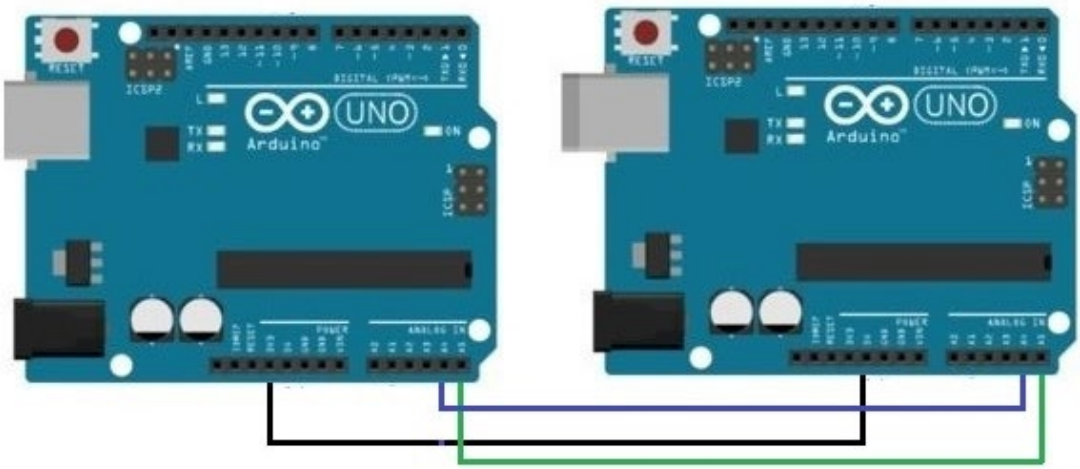

Figure 36. Arduino master/slave configuration.

The final item needed to autonomously run the robot and scanner would be a Raspberry Pi. The Raspberry Pi would be coded to run a Windows 10 Remote Desktop. The Raspberry Pi and the computer would communicate through the same Wi-Fi network. The Raspberry Pi was needed to process the information obtained from the scanner. By doing this the robot and scanner would have the processing power of a computer in a compact form.

RESULTS AND DESCUSSION

Overall the final system could run and avoid any obstacles in its path. The final assembly of the scanner onto the test robot can be seen in Figure 37. As mentioned before this will not be the final robot for the scanner. It is just a test bot to make sure the scanner and code work.

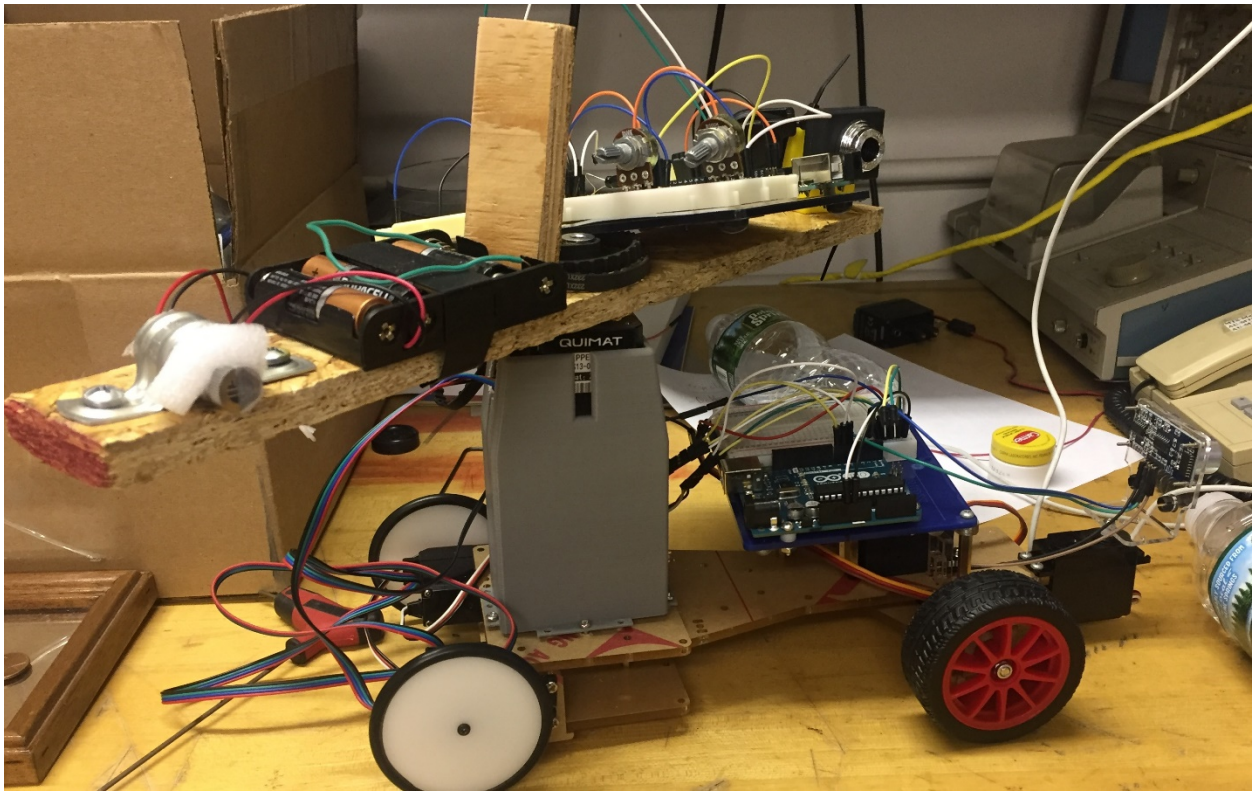

Figure 37. Final results.

This method used solve many of the problems used in other methods. For example, this method needed significantly less parts then many of the SLAM techniques discussed before. Compared to the RGB – D method this system needed a much smaller processing capability. Finally, when compared to both the set up could tell the distance of objects in the area through the use of an ultrasonic sensor, but in the other two methods discussed both would have to drive up to the obstacle and calculate the distance based off of its set speed and time it took to reach it. This method is not a problem free. The main problems that occurred after final assembly was scanner control and Raspberry Pi communication. As previously discussed the angle at which an area

scanned was preset before running. When the scanner was attached to the robot to run autonomously the angle set could not be adjusted. In most situations, this could be solved by keeping the scanner angle anywhere from 180 to 270°. The Raspberry Pi communication problem was due to the limited range of the Wi-Fi network that the Raspberry Pi had a remote desktop connection with. Possible solutions to this problem would be to extend the network range or increase the Raspberry Pi's Wi-Fi receiver.

CONCLUSION

When researching different 3-D mapping and obstacle avoidance techniques taken in the past there were many common flaws found throughout them. The three of these issues found in these methods were equipment size, power needs, and processing power. When designing and building the 3-D scanner for this project these problems were solved by using simple programs and controllers. One main concern in almost every method researched beforehand was the need to drive to the obstacle in the map generated by its system to calculate the distance of the object. To correct this problem in the 3-D mapping system created for this project, the ultrasonic sensor that was tested in the beginning was add to the system. By adding the ultrasonic sensor, the scanner could add the data from it to the map and determine the distances of the objects in the scanned area. The final step was combining the 3-D scanner with a land based robot to map the area drive through it autonomously and avoid all obstacles in its general path. This was done to test the capabilities of the scanner. When scanning an area from 0 through 270 degrees it took a relatively short time of approximately 20 seconds. Even though this system could function it still had a few issues like network connection. When using the scanner on a robot or alone it could generate a 3-D map of any area or object in a simple and time efficient manner. Overall the scanner will be able to assist the Bubblebot and many other robots to achieve autonomous capabilities.

The main goal of this project was to design, build, and program a 3-D laser scanner to be attached to the Bubblebot after its completion. The purpose of the Bubblebot will be to scan area, which would be hazardous for humans to be in, and navigate through any obstacles by fitting through any space. The hope for this project is that it would help in the protection of human life.

FUTURE WORK

Future work that can be done on this project could be changing the laser and web cam on the system. Currently, it's using a red line laser and a basic web cam to scan an area. This could be changed to an ultraviolet or infrared light and camera in order to reduce any ambient light errors that could occur in the scan. Other future work could include a better design for the robot system. The robot used in this project was very simple four wheels two motor generic design. By designing and creating a more advanced type robot the full potential of a scanner could be achieved. Currently a robot for this scanner is being designed by Wanrui He and Andres Garcia under the supervision of Dr. Lee at Alfred University. The main purpose of the robot is to expand and shrink to the desired size to fit through any space and avoid any obstacle seen by the scanner.

References

1. J. W. Powell, “The organization and plan of the United States Geological Survey,” *Am. J. Sci.,* **170** [29] 93-102 (1885).
2. W.F. Craven and J. L. Cate, *The Army Air Forces in World War II,* Vol. 6; Ch. 2 pp. 1-37. Univ. of Chicago Press, Chicago, 1955.
3. R. L. Hardy, “Theory and applications of the multiquadric-biharmonic method 20 years of discovery 1968–1988,” *Computers & Mathematics with Applications,* **19** [9] 163-208 (1990).
4. H. Liu, M. Philipose, and M. Sun, “Automatic objects segmentation with RGB-D cameras,” *J. Vis. Commun. Image R.,* **25** [4] 709–718 (2014).
5. P. Lourenço, P. Batista, P. Oliveira, C. Silvestre, and C.P. Chen, “Sensor-based globally exponentially stable range-only simultaneous localization and mapping,” *Robotics and Autonomous Systems,* **68** [7] 72–85 (2015).
6. H. Surmann, A. Nüchter, and J. Hertzberg, “An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments,” *Robotics and Autonomous Systems,* **45** [3] 181–198 (2003).
7. X. Gao, and T. Zhang, “Robust RGB-D simultaneous localization and mapping using planar point features,” *Robotics and Autonomous Systems,* **72** [2] 1–14 (2015).
8. D. Droeschel, M. Schwarz, and S. Behnke, “Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner,” *Robotics and Autonomous Systems,* **88** [6] 104–115 (2017).
9. B. Fry, and C. Reas, “Overview. A short introduction to the Processing software and projects from the community” (2008) Processing. Accessed on: April 2017. Available at <https://processing.org/copyright.html>
10. B. Fry, and C. Reas, “The History of Processing” (2008) Processing. Accessed on: April 2017. Available at <https://processing.org/overview/>
11. J. Liu, D. Liu, J. Cheng, and Y. Tang, “Conditional simultaneous localization and mapping: A robust visual SLAM system,” *Neurocomputing,* **145** [13] 269–284 (2014).
12. D. Rowan, “Kinect for Xbox 360: The inside story of Microsoft's secret 'Project Natal'” (2010) Wired Microsoft. Accessed on: September 2016. Available at <http://www.wired.co.uk/article/the-game-changer>

13. N. A. Othman and H. Ahmad, "Examining the eigenvalues effect to the computational cost in mobile root simultaneous localization and mapping," *Computers & Electrical Engineering,* **56** [2] 659-673 (2016).

14. S. Nasuriwong and P. Yuvapoositanon, "Posterior Elimination Fast Look-Ahead Rao-Blackwellized Particle Filtering for Simultaneous Localization and Mapping," *Procedia Computer Science*, **86** [11] 261–264 (2016).
15. C. Lai, and K. Su, "Development of an intelligent mobile robot localization system using Kinect RGB-D mapping and neural network," *Computers & Electrical Engineering,* (In Press, Corrected Proof).
16. S. Thrun, "Robotic Mapping: A Survey," *Natl. Sci. Found.,* **000** [7] 1-31 (2002).

Appendix A

Overall parts list and cost for the project

Part	Cost	Quantity	Subtotal
Stepper Motor	$11	1	$11
A4988 Motor Controller	$4	1	$4
Arduino Uno	$25	2	$50
Camera	$20	1	$20
Raspberry Pi	$40	1	$40
Micro SD Card	$10	1	$10
Line Laser	$25	1	$25
Ultrasonic Sensor HC-SR04	$3	1	$3
Servo	$13	3	$39
Misc.	$50		$50
Total			$252

Appendix B

This code is used by Arduino to control the ultrasonic sensor to determine the distances of the objects in the scanned area.

```
//Ultrasonic Range Finder


// Includes the Servo library
#include <Servo.h>.
// Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 10;
const int echoPin = 11;
// Variables for the duration and the distance
long duration;
int distance;
Servo myServo; // Creates a servo object for controlling the servo motor
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600);
  myServo.attach(12); // Defines on which pin is the servo motor attached
}
void loop() {
  // rotates the servo motor from 15 to 165 degrees
  for(int i=15;i<=165;i++){
  myServo.write(i);
```

```
  delay(30);

  distance = calculateDistance();// Calls a function for calculating the distance measured by the Ultrasonic sensor for each degree


  Serial.print(i); // Sends the current degree into the Serial Port

  Serial.print(","); // Sends addition character right next to the previous value needed later in the Processing IDE for indexing

  Serial.print(distance); // Sends the distance value into the Serial Port

  Serial.print("."); // Sends addition character right next to the previous value needed later in the Processing IDE for indexing

  }

  // Repeats the previous lines from 165 to 15 degrees

  for(int i=165;i>15;i--){

  myServo.write(i);

  delay(30);

  distance = calculateDistance();

  Serial.print(i);

  Serial.print(",");

  Serial.print(distance);

  Serial.print(".");

  }

}

// Function for calculating the distance measured by the Ultrasonic sensor

int calculateDistance(){


  digitalWrite(trigPin, LOW);
```

```
  delayMicroseconds(2);

  // Sets the trigPin on HIGH state for 10 micro seconds

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH); // Reads the echoPin, returns the sound wave travel time in
microseconds

  distance= duration*0.034/2;

  return distance;

}
```

Appendix C

This code is used by processing in order to allow the user to visualize a 2-D image of the scanned area and tell the distances of the objects.

```
//Processing Radar Generator

import processing.serial.*; // imports library for serial communication

import java.awt.event.KeyEvent; // imports library for reading the data from the serial port

import java.io.IOException;

Serial myPort; // defines Object Serial

// defubes variables

String angle = "";

String distance = "";

String data = "";

String noObject;

float pixsDistance;

int iAngle, iDistance;

int index1 = 0;

int index2 = 0;

PFont orcFont;

void setup() {


  size (1920, 1080); // ***CHANGE THIS TO YOUR SCREEN RESOLUTION***

  smooth();

  myPort = new Serial(this, "COM4", 9600); // starts the serial communication

  myPort.bufferUntil('.'); // reads the data from the serial port up to the character '.'. So actually it reads this: angle,distance.
```

```
  orcFont = loadFont("OCRAExtended-30.vlw");
}
void draw() {

  fill(98, 245, 31);
  textFont(orcFont);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0, 4);
  rect(0, 0, width, height - height * 0.065);

  fill(98, 245, 31); // green color
  // calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}
void serialEvent (Serial myPort) { // starts reading data from the Serial Port
  // reads the data from the Serial Port up to the character '.' and puts it into the String variable "data".
  data = myPort.readStringUntil('.');
  data = data.substring(0, data.length() - 1);

  index1 = data.indexOf(","); // find the character ',' and puts it into the variable "index1"
```

```
  angle = data.substring(0, index1); // read the data from position "0" to position of the variable index1 or thats the value of the angle the Arduino Board sent into the Serial Port
  distance = data.substring(index1 + 1, data.length()); // read the data from position "index1" to the end of the data pr thats the value of the distance


  // converts the String variables into Integer
  iAngle = int(angle);
  iDistance = int(distance);
}
void drawRadar() {
  pushMatrix();
  translate(width / 2, height - height * 0.074); // moves the starting coordinats to new location
  noFill();
  strokeWeight(2);
  stroke(98, 245, 31);
  // draws the arc lines
  arc(0, 0, (width - width * 0.0625), (width - width * 0.0625), PI, TWO_PI);
  arc(0, 0, (width - width * 0.27), (width - width * 0.27), PI, TWO_PI);
  arc(0, 0, (width - width * 0.479), (width - width * 0.479), PI, TWO_PI);
  arc(0, 0, (width - width * 0.687), (width - width * 0.687), PI, TWO_PI);
  // draws the angle lines
  line(-width / 2, 0, width / 2, 0);
  line(0, 0, (-width / 2)*cos(radians(30)), (-width / 2)*sin(radians(30)));
  line(0, 0, (-width / 2)*cos(radians(60)), (-width / 2)*sin(radians(60)));
  line(0, 0, (-width / 2)*cos(radians(90)), (-width / 2)*sin(radians(90)));
  line(0, 0, (-width / 2)*cos(radians(120)), (-width / 2)*sin(radians(120)));
```

```
  line(0, 0, (-width / 2)*cos(radians(150)), (-width / 2)*sin(radians(150)));
  line((-width / 2)*cos(radians(30)), 0, width / 2, 0);
  popMatrix();
}
void drawObject() {
  pushMatrix();
  translate(width / 2, height - height * 0.074); // moves the starting coordinats to new location
  strokeWeight(9);
  stroke(255, 10, 10); // red color
  pixsDistance = iDistance * ((height - height * 0.1666) * 0.025); // covers the distance from the sensor from cm to pixels
  // limiting the range to 40 cms
  if (iDistance < 40) {
    // draws the object according to the angle and the distance
    line(pixsDistance * cos(radians(iAngle)), -pixsDistance * sin(radians(iAngle)), (width - width * 0.505)*cos(radians(iAngle)), -(width - width * 0.505)*sin(radians(iAngle)));
  }
  popMatrix();
}
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30, 250, 60);
  translate(width / 2, height - height * 0.074); // moves the starting coordinats to new location
  line(0, 0, (height - height * 0.12)*cos(radians(iAngle)), -(height - height * 0.12)*sin(radians(iAngle))); // draws the line according to the angle
```

```
  popMatrix();
}
void drawText() { // draws the texts on the screen

  pushMatrix();
  if (iDistance > 40) {
    noObject = "Out of Range";
  }
  else {
    noObject = "In Range";
  }
  fill(0, 0, 0);
  noStroke();
  rect(0, height - height * 0.0648, width, height);
  fill(98, 245, 31);
  textSize(25);

  text("10cm", width - width * 0.3854, height - height * 0.0833);
  text("20cm", width - width * 0.281, height - height * 0.0833);
  text("30cm", width - width * 0.177, height - height * 0.0833);
  text("40cm", width - width * 0.0729, height - height * 0.0833);
  textSize(40);
  text("Object: " + noObject, width - width * 0.875, height - height * 0.0277);
  text("Angle: " + iAngle + " °", width - width * 0.48, height - height * 0.0277);
  text("Distance: ", width - width * 0.26, height - height * 0.0277);
```

```
  if (iDistance < 40) {
    text("        " + iDistance + " cm", width - width * 0.225, height - height * 0.0277);
  }
  textSize(25);
  fill(98, 245, 60);
  translate((width - width * 0.4994) + width / 2 * cos(radians(30)), (height - height * 0.0907) - width / 2 * sin(radians(30)));
  rotate(-radians(-60));
  text("30°", 0, 0);
  resetMatrix();
  translate((width - width * 0.503) + width / 2 * cos(radians(60)), (height - height * 0.0888) - width / 2 * sin(radians(60)));
  rotate(-radians(-30));
  text("60°", 0, 0);
  resetMatrix();
  translate((width - width * 0.507) + width / 2 * cos(radians(90)), (height - height * 0.0833) - width / 2 * sin(radians(90)));
  rotate(radians(0));
  text("90°", 0, 0);
  resetMatrix();
  translate(width - width * 0.513 + width / 2 * cos(radians(120)), (height - height * 0.07129) - width / 2 * sin(radians(120)));
  rotate(radians(-30));
  text("120°", 0, 0);
  resetMatrix();
  translate((width - width * 0.5104) + width / 2 * cos(radians(150)), (height - height * 0.0574) - width / 2 * sin(radians(150)));
```

```
  rotate(radians(-60));
  text("150°", 0, 0);
  popMatrix();
}
```

Appendix D

This code, used in Arduino, is used to test the stepper motor used to control the scanner.

```
//Stepper Motor and Driver Tester

// defines pins numbers
const int stepPin = 10;
const int dirPin = 9;

void setup() {
  // Sets the two pins as Outputs
  pinMode(stepPin,OUTPUT);
  pinMode(dirPin,OUTPUT);
}
void loop() {
  digitalWrite(dirPin,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 200; x++) {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
```

```
  digitalWrite(dirPin,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 400; x++) {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
}
```

Appendix E

This code, used in Arduino, is used to run stepper motor and set the minimum and maximum area the scanner will see.

```
//Stepper Motor Code for Scanner


#include <Wire.h>


int dir = 9;  // direction, clockwise or counter clockwise

int stp = 10;  // step command

int pot1 = 0; // speed

int pot2 = 1; // angleEnd


void setup(){

  Serial.begin(9600);


  pinMode(dir, OUTPUT);

  pinMode(stp, OUTPUT);

  pinMode(pot1, INPUT);

  pinMode(pot2, INPUT);


}


int del;  // delay between each loop, determines the speed of the motor

float angle = 0;  // the current angle is held in this
```

```
int angleEnd; // the furthest angle that the motor will turn to

int spd;  // the speed again, but just for displaying on the LCD screen


int pass = 0; // the number of times that the scanner has spun in each direction

int repeat = 2; // the number of times the scanner will turn in eech direction. Repeat of 2 returns it to it's starting position


void loop(){

  del = map(analogRead(pot1), 0, 1023, 500, 50);  // reading the pot1 value and mapping it to the delay

  spd = map(analogRead(pot1), 0, 1023, 1, 100); // reading the pot1 value and mapping it to the speed

  angleEnd = map(analogRead(pot2), 0, 1023, 0, 60);  //// reading the pot2 value and mapping it to the furthest angle


  if (pass < repeat){ // checking to see if the scanner needs to move

    if(pass%2 == 0){  // checking to see if the scanner has done an even number of passes

      digitalWrite(dir, LOW); // setting the steppers direction

      angle = angle + 1.8;  // increasing the current angle (quarter step)

    }

    else if(pass%2 == 1){ // checking to see if the scanner has done an odd number of passes

      digitalWrite(dir, HIGH);  // setting the steppers direction

      angle = angle - 1.8;  // decreasing the current angle (quarter step)

      Serial.println(500.0);  // telling processing to STOP after the first pass. Allows the scanner to return to it's starting position after one pass

    }
```

```
    digitalWrite(stp, HIGH);  // Start telling the stepper driver to take a step
    delayMicroseconds(300);   // tell it for a bit longer...
    digitalWrite(stp, LOW);   // stop telling it so take a step
  }

  if ((angle >= angleEnd || angle <=0) && pass < repeat){   // has the stepper motor gone beyond the specified final angle, or beyond it's starting point?
    pass++;   // if so, then it's completed a pass so increase the number of passes it's done
  }

  Serial.println(angle);    // tell Processing what angle the motor's at


  delay(del);   // wait for a bit before starting it all again, this determines the speed of the motor
}
```

Appendix F

This code, used by Processing, sets the web cam used to only record the red light from the laser, and allows Processing and Arduino to communicate

```
//Processing Camera Scanner


import processing.video.*;

import processing.serial.*;

Serial port;                // create a serial object called "port"

int linefeed = 10;          // ASCII character for new line


PrintWriter output;         // object for writing to the text file


Capture video;              // video capture object


PImage img1;                // first image with filtered red colour (thick fuzzy line)

PImage img2;                // second image which is a filtered version of the first image (thin, dotted line)


void setup(){

  size(640,960);                      // render window size, double the size of the webcam image to fit in filtered image too

  video = new Capture(this, 640, 480);    // specifying the webcam resolution

  video.start();                      // begining communication with the webcam

  port = new Serial(this, "COM4", 9600);  // setting up the serial communication with the Arduino, check correct COM port in Arduino - Tools - Port
```

```
  port.clear();                         // clear out the serial port before starting
  output = createWriter("test.txt");      // make the text file
}

String myString;     // String to hold angles from Arduino
float motangle = 0;  // holds the current motor angle

void draw(){
  img1 = createImage(640, 480, RGB);  // determining image resolution (same as webcam)
  img2 = createImage(640, 480, RGB);  // determining image resolution (same as webcam)
  background(0);                      // set black background of render window

  output.print(";");       // print a semi colon in the text file to indicate the start of a new frame
  output.print(motangle);  // print the motor angle in the text file

  if(video.available()){   // is there a new frame available from the webcam?
    video.read();          // if so, then read it!
  }

  if (port.available() >= 4){                  // is there 4 or more bytes in the serial buffer from the Arduino
    myString = port.readStringUntil(linefeed);   // if so, then read it until there is a new line and save it to "mystring" as a string
    if(myString !=null){                       // if it succesfully read something
      motangle = float(myString);              // angle converted from a string to a float and saved to "motangle"
```

```
  }
 }


 for (int i = 0; i < width*height/2; i++){  // loop to run through all of the webcam pixels
  if(red(video.pixels[i]) > 50){          // is the RED value in a pixel is MORE than this value (out of 255)
    img1.pixels[i] = color(255);          // then place a white pixel in image 1
  }
 }


 for (int i = 0; i < width*height/2; i++){  // run through all the pixels in image 1
  int k = 0;                              // reset "k" to zero


  while(brightness(img1.pixels[i]) == 255 && i < (width*height/2)-1 && (i/width)%5 == 0){  // stay in this loop IF: there is a white pixel in image 1
                                                                                          //AND if it's not run off the edge of the image
                                                                                          //AND only if it's on every 5th row
    k++;  // increase "k" by 1. This counts the number of white pixels in a consecutive row
    i++;  // increase "i" by 1. moves along to next pixel
  }


  if(k > 0 && !(i-(k/2) == 195201)){    // check if "k" found any white pixels AND ignore a specific dead pixel (you can delete this bit)
    img2.pixels[i-(k/2)] = color(255);  // draw a white pixel in image 2 at the middle of a row of white pixels from image 1
```

```
    output.print(",");              // print a comma to the text file to seperate from motor angle (already printed)
    output.print(i%640);             // print the columb the new white pixel is on to the text file (X-coordinate)
    output.print(",");              // print a comma to the text file to seperate the coordinates
    output.print(i/640);             // print the row the new white pixel is on to the text file (Y-coordinate)
   }
  }

  image(video,0,0);   // show the webcam feed in the top of the render window
  image(img2,0,480);  // show image 2 in the bottom of the render window

  println(motangle);  // prints the current motor angle to the command window

  if (motangle > 450.0){  // if the motor angle from the Arduino is more than 450 deg, i.e. 500 deg, then stop the program
    output.flush();       // finish printing to the text file
    output.close();       // close the text file
    exit();               // stop this program from running
  }
}

void keyPressed(){      // make this program able to stop with any key
  output.flush();       // finish printing to the text file
  output.close();       // close the text file
  exit();               // stop this program from running
```

Appendix G

This code, used by Processing, uses the text file and all the data obtained in the scan to generate a 3-D map for the user to visualize what the scanner and robot see.

```
//Processing 3D Image Generator


void setup(){
  size(1000, 1000, P3D);  // make the render window 1000 x 1000 pixels and 3D capable
}


float rotmot;              // current angle of the motor
float rotcam = 30*PI/180;    // CHANGE THIS ACCORDINGLY the angle of the webcam where 0deg is straight forward and 90deg is looking at the laser (converted to radians)
float rotlaser = 0;         // same as above but for the linear laser
float rotpixX;              // angle of the current white pixel from the camera's centre view in X-direction
float rotpixY;              // angle of the current white pixel from the camera's centre view in Y-direction


float pixangleX = 50.0/640*PI/180;  // angle between each pixel in webcam in X-direction
float pixangleY = 50.0/640*PI/180;  // angle between each pixel in webcam in Y-direction


float[] A,B,AB;              // constants needed for line equation
float[] C,D,CD;              // constants needed for plane equation
float n,t;                  // constants needed for line/plane intersection equation
float pointX, pointY, pointZ;  // temporarily holds the X,Y,Z coordinates of a new point in 3D space
```

```
float[] drawpointX;          // holds all X coordinates of points in 3D space
float[] drawpointY;          // holds all Y coordinates of points in 3D space
float[] drawpointZ;          // holds all Z coordinates of points in 3D space

int k = 0;  // counts the number of white pixels in entire scan for placing them in the "drawpoint" vectors

void draw(){
  background(50);                          // make the background black
  perspective(PI/3.0,(float)width/height,1,500);  // adjust the near and far clipping planes to see the whole scene (last two number)
  stroke(255, 0, 0);                       // make the line red
  line(0,0,0,50,0,0);                      // make a line in the x-direction of 50
  stroke(0, 255, 0);                       // make the line green
  line(0,0,0,0,50,0);                      // make a line in the y-direction of 50
  stroke(0, 0, 255);                       // make the line blue
  line(0,0,0,0,0,50);                      // make a line in the z-direction of 50

  camera(50*cos(mouseX/100.0)+10, 20, -50*sin(mouseX/100.0)-30,  // X,Y,Z location of the camera (not the webcam)
       10, 0, -30,                                  // X,Y,Z location of what the camera is looking at (not the webcam)
       0.0, -1.0, 0.0);                             // where is up for the camera (not the webcam)

  if(keyPressed){  // only renders point cloud if a key is pressed on the keyboard
```

```
  String[] myString = loadStrings("test.txt");                    // save the text file to the first element of a string vector

  drawpointX = new float[splitTokens(myString[0], ",").length/2];  // make the drawpointX vector the length of how many white pixels there are

  drawpointY = new float[splitTokens(myString[0], ",").length/2];  // make the drawpointY vector the length of how many white pixels there are

  drawpointZ = new float[splitTokens(myString[0], ",").length/2];  // make the drawpointZ vector the length of how many white pixels there are

  myString = splitTokens(myString[0], ";");                       // split up the large string into multiple string each containing a webcam frame


  k = 0;  // reset "k" to zero


  for(int i = 0; i < myString.length; i++){              // repeat loop for every element in "myString" (number of webcam frames)

    String[] stringPart = splitTokens(myString[i], ",");  // split "myString" up into each pixel coordinate

    rotmot = float(stringPart[0]);                      // save the motor angle from the first string element to "rotmot"


    pushMatrix();            // create set of local coordinates

    rotateY(-rotmot*PI/180);  // rotate everything in Y by the motor angle


    pushMatrix();            // create another set of local coordinates

    translate(-19, 2, 0);     // move everything along the scanner arm to the laser position in 3D space

    rotateY(rotlaser);        // rotate everything in Y by the laser angle
```

```
//Plane Equation

C = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};  // create a point on laser plane with global coordinates

translate(10, 0, 0);                                  // move along by 10 in X-direction

D = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};  // create another point, 10 in X-direction away from laser plane

CD = new float[]{D[0]-C[0], D[1]-C[1], D[2]-C[2]};            // create a line perpindicular to plane using points "C" and "D" (NORMAL VECTOR)

// CD[0]x + CD[1]y + CD[2]z = n


popMatrix();  // close off local coordinates to return current position to the motor shaft with the motor's angle


for(int j = 1; j < stringPart.length; j = j+2){      // make the loop repeat for every second element in the "stringPart" vector (repeats for every set of X-Y white pixel coordinate in a single webcam frame)

    rotpixX = (float(stringPart[j])-320)*pixangleX;    // measuring the white pixels angle in X-direction from the cameras centre

    rotpixY = (240-float(stringPart[j+1]))*pixangleY;  // measuring the white pixels angle in Y-direction from the cameras centre


    pushMatrix();          // move into another set of local coordinates

    translate(18, 4.5, 0);  // move along the arm from the motor shaft to the webcam's centre

    rotateY(rotcam);       // rotate to the webcam's angle

    rotateX(rotpixY);      // rotate the imaginary line by "rotpixY" in the X-axis to align with current white pixel

    rotateY(rotpixX);      // rotate the imaginary line by "rotpixX" in the Y-axis to align with current white pixel
```

```
//Line Equation

A = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};  // create a point on webcam's centre with global coordinates

translate(0,0,10);                                  // move along by 10 in Z-direction

B = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};  // create another point, 10 in Z-direction out in front of the webcam, along the imaginary line

AB = new float[]{B[0]-A[0], B[1]-A[1], B[2]-A[2]};            // create the line from the webcam using points "A" and "B" (PROJECTION LINE)


popMatrix();  // close off local coordinates to return current position to the motor shaft with the motor's angle


// Intersection of Line and Plane

t = (n - CD[0]*A[0] - CD[1]*A[1] - CD[2]*A[2])

/   (CD[0]*AB[0] + CD[1]*AB[1] + CD[2]*AB[2]);  // Equation for the laser plane rearranged with equations for projection line substituted in to find constant "t"

pointX = A[0] + AB[0]*t;                    // Using "t" in line equation to find X-coordinate of intersection

pointY = A[1] + AB[1]*t;                    // Using "t" in line equation to find X-coordinate of intersection

pointZ = A[2] + AB[2]*t;                    // Using "t" in line equation to find X-coordinate of intersection


drawpointX[k] = (pointX);  // inserting temporary X-coordinate into the X-coordinate vector to hold all data points

drawpointY[k] = (pointY);  // inserting temporary Y-coordinate into the Y-coordinate vector to hold all data points

drawpointZ[k] = (pointZ);  // inserting temporary Z-coordinate into the Z-coordinate vector to hold all data points
```

```
      k++;                    // increment "k" for moving onto the next white pixel i.e. data point
    }
    popMatrix();  // close off local coordinates to return current position to (0,0,0) in global space
   }
  }

 if(k > 0){                                  // check to see if any data points were created
  for (int i = 0; i < drawpointX.length; i++){          // repeat loop for every data point
    stroke(255);                                 // make the data point white
    point(drawpointX[i], drawpointY[i], drawpointZ[i]);  // draw the point cloud using the
"drawpointX,Y,Z" vectors
   }
  }
}
```