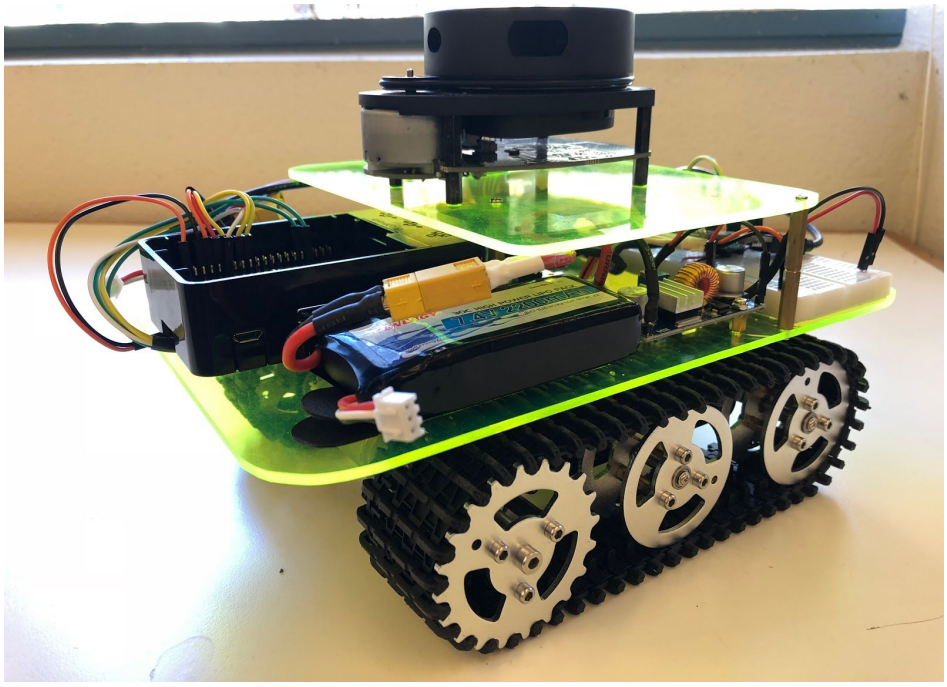

Autonomous Navigation and Mapping using LiDAR

Steven Alsalamy: alsalamy.steven@gmail.com
Ben Foo: benjamincfoo@gmail.com
Garrett Frels: gfrels18@gmail.com

Faculty Advisor: Dr. Andrew Danowitz

Computer Engineering Department
California Polytechnic State University, San Luis Obispo
June 2018

Table of Contents

Introduction

Project Overview:

The goal of this project was to make a fully autonomous robot, capable of mapping its surroundings and navigating through obstacles. This was done through the use of a chassis fitted with tracks and two motors, a lidar, a compass, and a Raspberry Pi. The robot also contains two batteries and is self powered. Encoders are used on the motors in order to track the distance traveled for more precise mapping and movements.

Potential Applications:

There are many different potential applications for which this project could be used as a base. The underlying concept is that any type of utility vehicle can be equipped with the same devices used in this project to move autonomously in a confined space.

Some examples:

- Autonomous Vacuum/Mop
- Autonomous Lawn Mower
- Autonomous Forklift

Project Goals and Objectives

The goal of this project is to create a fully autonomous robot. Our first objective is to use a LiDAR to scan all immediate surroundings. Multiple scans will then be used to build a two dimensional map of an entire area. The second objective is to be able to precisely navigate the map through use of the motors and encoders : this includes being able to move forward, rotate in place, and measure distance traveled. The third objective is to integrate an orientation sensor to

assist with precisely turning the robot, and maintaining the current orientation. The fourth objective is to implement a path planning algorithm which, given a scan of the current surroundings, is capable of planning a path for the robot to follow that avoids all obstacles. With all of these objectives completed, our overall goal of an autonomous robot will be possible.

Project Outcomes and Deliverables

- Construct the physical robot to be completely mobile
- Create a map using input from the LiDAR
- Implement precise motor controls for the robot
- Use the map to construct an obstacle free path

Background

Projects similar to this have been done in the past, and using LiDAR to scan areas has become a popular concept. Companies like Google, Uber, BMW, and Audi are all currently working to make their own autonomous vehicles. While these projects are on a much larger scale, they are similar to ours in that they use LiDAR to scan their surroundings and make their movements accordingly. They are also similar because the purpose of both is to navigate their surroundings while avoiding obstacles, even moving ones.

Engineering Specifications

To accomplish our goal, the following engineering specifications in Table 1 were created. A minimum scanning range of 5 meters was chosen so that a well informed path could be

constructed after every scan. A mapping area of 20,000 square meters is large enough to scan almost any building, and with a 20mm scale any significant objects will be detected. The chassis movement requirements were set to make sure that the robot does not veer off course to the point where it begins to affect the construction of the map. The robot needs to be small enough to maneuver through tight spots in a building. Scanning an entire building should never take more than two hours, and the batteries should be able to power the robot for that entire duration.

The table contains several parts. The tolerance section is used to indicate minimum or maximum requirements for each specification. The risk section describes the level of difficulty for meeting the corresponding engineering specification. Lastly, the compliance section of the table is used to indicate how each design requirement is to be met: analysis (A), test (T), and similarity to existing designs (S).

Spec. #	Parameter Description	Requirement	Tolerance	Risk	Compliance
1	Minimum LiDAR scan range	5 meters	500mm	low	A, T
2	LiDAR scan speed	Should take less than 1.5 seconds to for a complete 360 degree scan	250ms	high	A, T
3	Minimum mapping area	20,000 square meters	5,000 square meters	low	A, T
4	Minimum map scale	Each index in the map should represent 20 square millimeters of real space	30mm	low	A, T
5	Chassis rotation	Accuracy within ±1°	0.250°	high	A, T

	precision				
6	Chassis linear distance precision	Accuracy within ±10mm	5mm	high	A, T
7	Dimensions of entire robot	10” W x 10” L x 10” H	2” in W and L	low	A
8	Maximum weight of entire robot	5 lb	2 lb	medium	A
9	Minimum operating time on single charge	120 Min	0 min	low	A, T

Table 1. System specifications and requirements

Design Development

Assembling the Robot

We decided to use a continuous track, also known as tank tread, for this project. This was done to achieve higher traction to make movements as precise as possible. Each track is operated by its own individual motor and encoder. To make the robot as compact as possible, we created two separate seperate platforms to mount things on, with the motors mounted below everything. The lower platform, as seen in Figure 2 in Appendix D, contains the Raspberry Pi, two LiPo batteries, two voltage regulators, two electronic speed controllers, the BNO055 absolute orientation sensor, and a breadboard. On the top platform, as seen in Figure 3 in Appendix D, the LiDAR is mounted by itself to ensure that nothing obstructs the two dimensional plane in which it scans. To make sure that the LiDAR and compass do not shift in position when the robot rotates, they have been mounted on the center of the chassis. To also make sure that the robot does not veer to one side when moving, everything has been mounted in a symmetrical fashion. The LiPo batteries used are each 7.4V 2200mAh. Both voltage regulators are used to step down

voltage to 5V. One battery will provide power to the Raspberry Pi, which in turn provides power to the LiDAR and absolute orientation sensor. The second battery will provide power to the motors. For information on power consumption refer to Table 2 in Appendix A. The wiring diagram for the entire system can be seen in Figure 1 Appendix C, and will be discussed in more detail in the following sections.

LiDAR

The RPLIDAR A1M8 uses a UART to serial USB converter to communicate to the Raspberry Pi[1]. Using a Raspberry Pi allows us to take advantage of the USB interface functionality provided by the Raspbian operating system. Communicating with serial devices, is as simple as opening, reading, and writing to a file; the file handle /dev/ttyUSB0 acts as a buffer for the LiDAR. The LiDAR accepts several commands that allow us to start and stop a scan, get LiDAR health information, and restart the LiDAR. When scanning, the LiDAR sends multiple packets of data each containing the distance, angle, and quality of a scan. Data to and from the LiDAR comes in several types of packets with different formats. To quickly construct these packets when reading and writing data, we created a C struct for each packet format. The motor that controls rotation of the LiDAR motor, is powered on and off by disabling and enabling the USB DTR control signal.

Mapping

Information about the robot's surroundings is stored in a two dimensional array that will serve as a grid. Each index in the array will be a single byte, allowing us to mark each index as

either empty, an obstacle, or not scanned. Each index represents a square space in the robot's real environment. The size of the square in each index is dependant upon a scale that we have defined in a header file and is easily adjustable; for higher resolution maps we use a smaller scale. All distance measurements in this project are done in millimeters. A 20mm scale gives us enough resolution for movement and detecting small objects. The size of the map is also easily adjustable through a definition in C. The two dimensional array that represents the entire map is declared as map[MAP_SIZE][MAP_SIZE], where MAP_SIZE is defined as ten times double the max range the LiDAR can scan, divided by the scale of each index. For example 10 * (24000 / 20), where 24000 is the double the max range, 20 is our scale. A multiple of ten is used to give us a large enough map of 57,600 meters squared, while only using about 10% of memory; theoretically we can use a multiple of 25 to use up almost all of the RAM memory, and generate a map that is 360,000 meters squared in area. Double the range of the LiDAR is used to account for the fact that we initialize the location of the robot to be at the center of the map. Indexes into the two dimensional array are done in an X,Y fashion(map[y][x]), where index (0,0) is the farthest north-west index, (max,max) is the farthest south-east index, and (max/2,max/2) is the center of the map. Refer to figure 4 in the appendix E for a visual depiction of the map.

Motors

For movement, two brushless DC motors are used. With a 1:75 gear ratio, these motors have enough torque to easily move the chassis. In addition to the built-in gear boxes, these motors also have hall effect encoders that are used to calculate linear distance traveled. To easily control speed and direction of these motors, two TI DRV8838 electronic speed controllers were

used. By supplying a pulse width modulation signal as an input to the DRV8838, a proportional DC output voltage can be sent to the motors to control speed. The DRV8838 also has a Phase input which toggles the direction that the motor will spin. The Raspberry Pi supplies two individual PWM signals at 1.92kHz, one for each electronic speed controller. Due to how the Raspberry Pi generates PWM signals, this relatively low frequency allows for a duty cycle resolution of 100 steps [5]. In other words, the duty cycle of this signal can be varied from 0% to 100% in steps of 1%. This variable duty cycle PWM signal allows for a variable DC output from the DRV8838's which will maintain that 100 step resolution [6]. For example, an input signal to the DRV8838 with a duty cycle of 75% will output a DC voltage that is 75% of the motor voltage supply (VIN, or pin 7, on the DRV8838). It is also set to supply two individual GPIO outputs which are tied to the phase input of the DRV8838's. This setup allows each motor to operate independently of each other allowing for precise calibration of movement and turning. To keep track of linear distance, the hall sensors on the motors are used to trigger interrupts on the Raspberry Pi. To make accurate turns, the Bosch BNO055 absolute orientation sensor is used[2].

Orientation Sensor

In order to keep track of the robots orientation at all times we decided to the use the Bosch BNO055 absolute orientation sensor[2]. The sensor has been integrated by Adafruit onto a breakout board with a driver library that communicates over UART for using it with a Raspberry Pi. However, because the driver library is written in Python, and our project is being implemented in C, a wrapper file had to be written to embed the python module in C[3]. For the

orientation sensor to function correctly it must first be manually calibrated by continuously moving and rotating it for a short period of time. The calibration settings can be saved and then loaded onto the sensor each time it is powered on. The system has been implemented in a way to set the initial direction the robot is facing when powered on as its relative North; this was done because the orientation sensor does not always return absolute North. The angle in which the robot is facing increases from 0 to 360 in a clockwise fashion.

Path Planning

To determine the shortest path from point A to B while avoiding obstacles, we use the A-Star algorithm[4]. The A-Star algorithm is an informed search algorithm that uses the cost of each movement to decide which path to take. The cost of moving to a specific spot is calculated by summing up the number of movements made to get to your current location, and a heuristic that calculates the shortest distance remaining to get to the destination. In our case the heuristic is calculated as abs(current x - end x) + abs(current y - end y). Before considering moving to the next spot, the algorithm first checks if that index on the map contains an obstacle. Currently the algorithm has been implemented to make horizontal and vertical movements on the map, but can be enhanced later on to also check for diagonal movements. The algorithm maintains two lists that keep track of all possible next movements, and all movements that have already been attempted. The data structures used for maintaining the two lists are a binary minimum heap, and a regular array. The speed in which the algorithm can find a path is heavily reliant on how well these data structures can add new elements, search for an element, and find the lowest cost element.

Final Design Concept

Creating separate drivers for each peripheral gives us a very modular design, making it much easier to test the whole system and integrate everything together. This also allows us to replace each individual peripheral with a different model if necessary.

In our final system design we start by populating the map with information scanned in from the LiDAR. To do this we first populate a much smaller grid that will be later merged into the over all map; this is done to allow us to implement a multithreaded system in the future. The initial grid is just big enough to store the robots immediate environment that is with in range of the LiDAR. A single 360 degree scan from the LiDAR is first requested. Using a combination of the distance and angle of each scan, along with the current location of the robot(being just the center of the initial smaller grid we populate), we can use simple trigonometry to determine the location of each object in the robots environment. The angle used is a combination of the angle returned from the LiDAR and the direction of the robot measured from the compass in order to determine what the true angle is relative to the robot. To account for if an object moves we also set each index between the scanned object and the robot as empty. Using the actual location of the robot on the map, we can overwrite that section of the map with the newly populated smaller grid.

Each time the map is updated, the robot then attempts to calculate a path from its current location to whatever the destination may be; currently this is provided by the user to the system. If a path exists, a list of x,y coordinates on the map are returned that represent the path from start to finish. This list is then translated into a series of rotations and forward movements. Using the direction that the robot is currently facing, scanned from the compass, along with the current and

next index it must move to, we can determine how it must rotate. The motors then rotate the robot accordingly and move forward the distance we have set as our scale. If the robot is to avoid moving obstacles, it must update the map after just a single movement. However if operating in an environment with stationary obstacles, the robot can continue processing movements until an index in the map that has not been scanned has been reached.

Future Work

Overall system

The best way to improve the current system without having to make any hardware changes would be to implement a multithreaded system. The Raspberry Pi Model 3 B+ is a quad core computer. To use the Raspberry Pi to its fullest extent, it would be most efficient to create three threads. A thread for scanning data from the LiDAR and populating the grid, another for running the path planning algorithm, and the last one for motor control.

LiDAR

With a multithreaded system we can continuously read scan data from the LiDAR in the form of a stream. This will immediately remove the large amount time required to clear the USB serial buffer each time before starting a new scan. The current LiDAR being used also has trouble sometimes getting accurate scans from dark objects that do not reflect light very well. With a higher spending limit a better LiDAR can be used for more accurate results and faster scans.

Map

Currently the only way to view the map is by printing the two dimensional array to a file. This is great for testing and development, however as the map gets larger it becomes very difficult to see everything at the same time. Formatting the data in a way that can be accepted by a two dimensional grid plotter would allow us to much more clearly view the plane that was scanned.

Path Planning

As discussed in the development section for the A-Star algorithm, the time taken calculate a new path can be decreased by using a more efficient data structure for the lists. Along with using a min heap, a hash table can also be implemented to decrease search time for specific elements to O(1). Another great way to calculate a path faster would be to flag indexes on the grid as parts of the path. Instead of calculating a new path each time the map is updated, we can instead only calculate a new path when an obstacle obstructs the old path. We would then also only have to recalculate a small portion of the path instead of an entirely new path.

Motor Control

Our system is currently heavily reliant on precise movement from the motors to keep track of the robots exact location. Although we have calibrated the motors we are currently using to be as accurate as possible, we are still have issues with making sure both motors rotate at the same rate. We have identified this to be an issue internally with the motors, and the best action would be to purchase higher quality motors that can provide the accuracy we need.

Conclusion

The purpose of this project was to show how LiDAR could be used to autonomously map and navigate an area. The system implemented in this project can serve as a base model for how commercial equipment like an autonomous vacuum or forklift could be implemented. Although the current design is limited to lower quality parts, we were still able to achieve very accurate maps and accurately move the robot with in the map. We hope in the near future to have implemented a way for the robot to move more freely than just making 90 degree turns, and to make the system multithreaded in order to decrease the amount of time it takes to scan a room and calculate a path. Over all, we feel that this project has been a great culmination of everything we have learned while pursuing our computer engineering degrees.

References

[1]T. Huang, "SLAMWARE - Slamtec - Leading Service Robot Localization and Navigation Solution Provider", *Slamtec.com*, 2018. [Online]. Available: http://www.slamtec.com/en/Support#rplidar-a1. [Accessed: 12- Jun- 2018].

[2]T. DiCola, "Overview | BNO055 Absolute Orientation Sensor with Raspberry Pi & BeagleBone Black | Adafruit Learning System", *Learn.adafruit.com*, 2018. [Online]. Available: https://learn.adafruit.com/bno055-absolute-orientation-sensor-with-raspberry-pi-and-beaglebone-black/overview. [Accessed: 12- Jun- 2018].

[3]"Python/C API Reference Manual — Python 2.7.15 documentation", *Docs.python.org*, 2018. [Online]. Available: https://docs.python.org/2/c-api/index.html. [Accessed: 12- Jun- 2018].

[4]"A* pathfinding algorithm", *Growing with the Web*, 2018. [Online]. Available: http://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html. [Accessed: 12- Jun- 2018].

[5]"WiringPi", *Wiringpi.com*, 2018. [Online]. Available: http://wiringpi.com/. [Accessed: 12- Jun- 2018].

[6]"Pololu - DRV8838 Single Brushed DC Motor Driver Carrier", *Pololu.com*, 2018. [Online]. Available: https://www.pololu.com/product/2990. [Accessed: 12- Jun- 2018].

Appendices

Appendix A: *Power Consumption*

	Device	Avg Current Draw (mAh)	Max Current Draw (mAh)	Voltage Range (V)
1	Raspberry Pi	700	1000	4.9-5.5
2	RPLIDAR A1M8	400	700	4.9-5.5
3	Bosch Absolute Orientation Sensor	--	12.3	2.4-3.6
4	Motors	600	1200	0-9
5	TI DRV8838	0.7	1.5	1.8-7

Table 2. Power consumption of entire system determined from data sheets of each device

Appendix B: *Bill of Materials*

Item #	Part Name	Part Number	Price	Quantity	Extended Price
1	Raspberry Pi 3	RB-Ras-12	39.95	1	39.95
2	LiDAR	RPLDIAR-A1M8	199.00	1	199.00
4	Orientation Sensor	BNO055 9 DOF	34.95	1	34.95
5	Electronic Speed Controller	TI DRV8838	3.49	2	6.98
6	Standoff screws	M2.5	9.99	1	9.99
7	Tank Chassis		49.99	1	49.99
8	LiPo Battery	Tenergy 7.4V 2200mAh	14.99	2	29.98
9	Acrylic base		9.99	1	9.99
10	Voltage Regulator		3.49	2	6.98
				Total	387.81

Table 3. Bill of Materials(Not including equipment for manufacturing)

Appendix C: *Wiring Diagram*

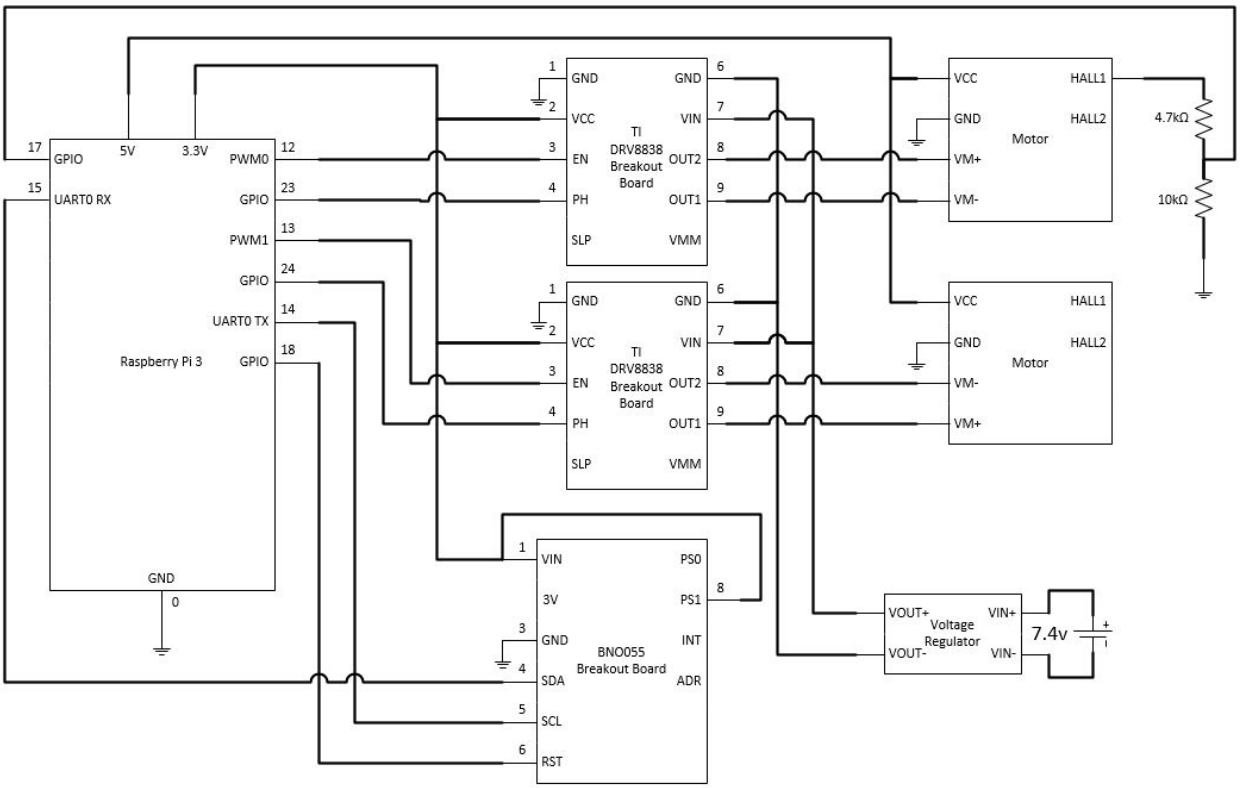

Figure 1. Wiring Diagram for Raspberry Pi, BNO055 Sensor, Motors, ESC's, and voltage regulators

Appendix D: *Robot Structure*

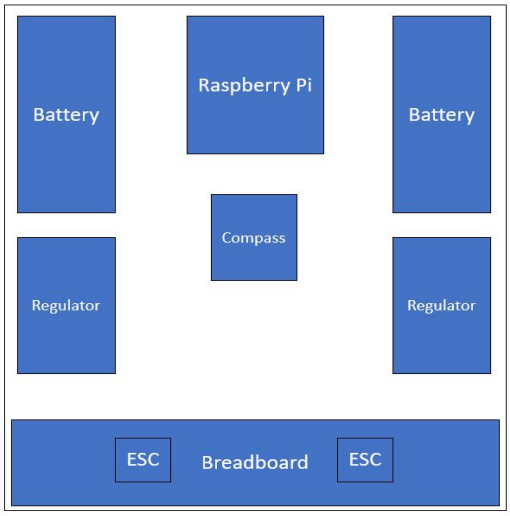

Figure 2. Lower Level of Robot

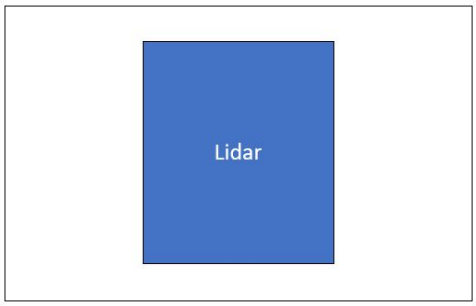

Figure 3. Top Level of Robot

Appendix E: *Map*

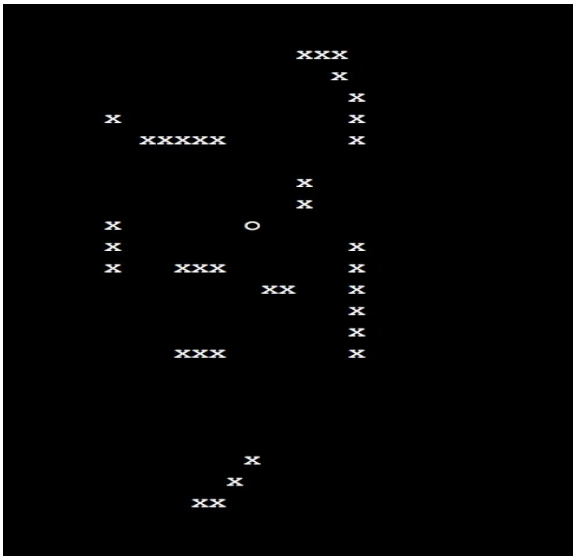

Figure 4. Map, O: robot, X: Obstacle Note: Resolution has been decreased in order to fit on screen

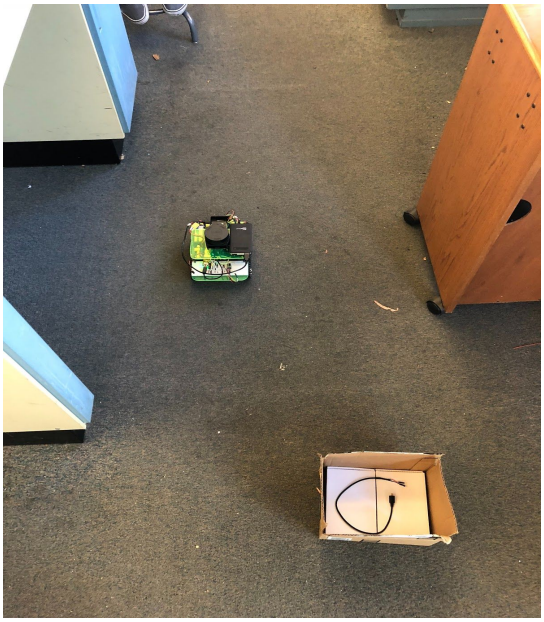

Figure 5. Actual surroundings