Edge Detection by Centimeter Scale Low-Cost Mobile Robots

Erin Wong

CMU-RI-TR-23-12

April 25, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Zeynep Temel, *chair*
Wennie Tabib
Brady Moon

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my parents who have always believed in me. It is because of their support that I have gotten to where I am today.

Abstract

In Search and Rescue (SaR) efforts after natural disasters like earthquakes, the primary focus is to find and rescue people in building rubble. These rescue efforts could put first responders at risk and are slow due to the unstable nature of the environment. Robotic solutions capable of gathering information are useful because they can provide environment information to the first responders that decrease the risks of rescue efforts. However, navigating a post-disaster environment is an unsafe task by nature because depending on the size and capability of the robots, they may get stuck, lost, or damaged in unpredictable situations, which can be very costly when using expensive robots.

With this thesis, we present the implementation of two low cost sensors, i.e., a photoresistor and an ultrasonic sensor, onto an existing low-cost, small-scale mobile robotic system. We implement these sensors to give the robot the ability to navigate unknown environments and map the gaps. The low-cost nature makes these robots an ideal candidate for navigating near edges – even in the event that the robot falls off, there is little monetary loss. After presenting mechanical and electrical changes needed to accommodate these sensors onto the robot, we present three algorithms to detect an edge, that perform with varying safety levels and mapping capabilities. We then demonstrate the results of these algorithms in simulation environments where the robot is traversing a convex platform. These algorithms successfully gather new information about the platform and provide maps that would make future traversal of the environment safer.

Acknowledgments

This work would not exist if not for the incredible support, care, work, and love of an army of people. I cannot thank each of them enough.

First and foremost, I would like to thank my advisor, Zeynep Temel, who has guided and supported me in the past two years to help me achieve my goals.

Thank you also to my wonderful committee, Wennie Tabib and Brady Moon for taking time out of their very busy schedule to help me, give me feedback, and be on my committee.

I could not have done this work without all the guidance from Yisha, the original PuzzleBots creator. She taught me so much about the hardware and helped me understand the overall system , which was fundamental to the research presented in this thesis. I owe her a million thank yous.

Thank you to everyone else in Zoom Lab, for your support, friendship, and laughter. I will miss you all dearly.

Thank you to all my other friends, both in and out of CMU who have kept me sane even in the most difficult times. Even among all the work of grad school, you have helped me create memories that I will keep forever.

And finally, thank you to my parents who were always there for me and never wavered in their belief in me, even when I doubted myself. They brought me to where I am today, and I will never be able to thank them enough.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Search and Rescue (SaR) is a prime challenge suited for robotic solutions because robots can navigate affected areas after natural disasters like earthquakes or tornadoes, in order to protect first responders from life-endangering situations. Even in the event of robots not being able to save survivors on their own, they could provide critical information about the environment that could help locate survivors, and make it easier, faster, and safer for first responders to rescue people in unknown environments. As an extension, multi-agent robotic systems are also an ideal solution for SaR tasks because of their ability to cover a large area at once.

Mapping the new post-disaster environment is the key to finding as many survivors as possible. However, even if previous building floor plans are available, a lot of the area could still be unknown due to the effects of the disaster – an earthquake could have collapsed parts of floors, rendering some areas unreachable by simple locomotion, and requiring more complex gap crossing solutions. Giving the robots the ability to cross gaps in disaster situations, expands the explorable area with robotic solutions.

The Puzzlebots platform as originally formulated in [31, 32] is a structurally compliant, small-scale robotic swarm system. Each robot measures approximately 5 cm in each dimension and is equipped with a locking mechanism that allows robots to couple and decouple with other robots in the swarm. This allows the swarm of robots to work together to build bigger structures than their size, which makes them

able to cross gaps between platforms.

A key characteristic of the Puzzlebots platform is that they are low-cost to build, which is especially important for scalability as they are designed to work as a swarm, with multiple other agents. The main chassis is 3D-printed Thermoplastic Polyurethane (TPU) and all the electronic components can be easily purchased off-the-shelf for a low cost. In SaR, having a multi-agent system where each unit is low-cost is especially appealing because having expensive technology can cause an inequitable response in SaR efforts in disadvantaged communities [6]. Using a low-cost robot in SaR also allows a robot to be lost in the field, which could happen when exploring unknown environments, without worrying about monetary loss.

In this thesis, we present hardware and software additions to the original Puzzlebots platform that allow the robots to autonomously detect and map gaps in their environment.

1.2 Problem Definition

When navigating an unknown environment, there may be gaps that robots can fall into. PuzzleBots [31, 32] is a swarm robotic system, where robots can work together to create configurations capable of crossing gaps larger than the size of one Puzzlebot. However, the original Puzzlebots system had no sensors equipped, and thus no ability to detect a gap, avoid it, or decide on its own how many robots were needed to traverse a gap. Adding sensors, gap mapping, and a gap avoidance algorithm is a clear first step to bringing these robots towards autonomy. We aim to demonstrate this functionality with PuzzleBots on a convex platform.

1.3 Relevant Works

1.3.1 Puzzlebots Background

The Puzzlebots platform was originally proposed by Yi et al. in [31] as a swarm robotic system capable of performing coupling and decoupling behaviors without extra actuation beyond locomotion. Previous modular robotic work also aimed to build coupling and decoupling robotic modules, such as ATRON[5], M-TRAN III[33],

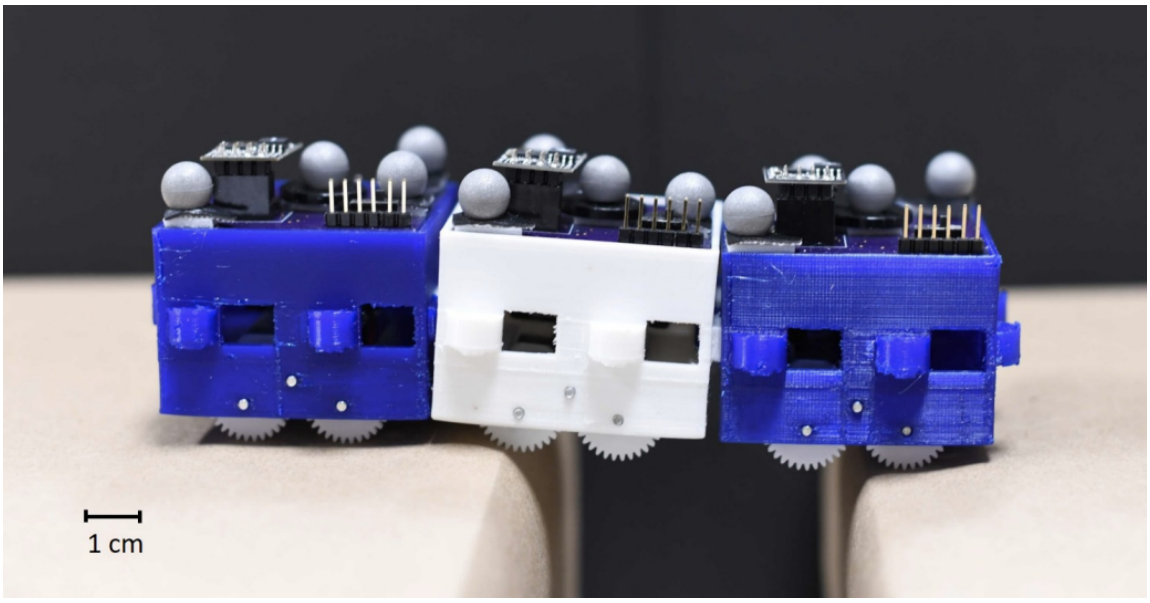


Figure 1.1: Three PuzzleBots are configured together in order to cross a gap. This image is taken from [31].

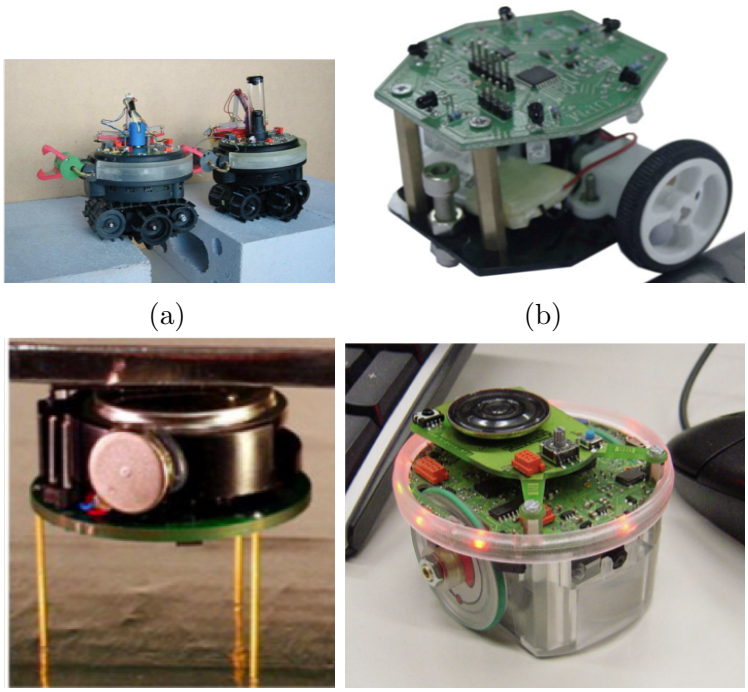
SlimeBot[24], Lily[11], M-blocks[22], SMORES[16], FreeBot[15], and Swarm-bot[8]. These systems performed coupling and decoupling in a variety of methods such as magnetic force, fluid actuation, and individual grippers. These methods can be expensive, high in power consumption, or are only able to bare very limited loads. The original goal of Puzzlebots was to address these issues in previous modular solutions [31].

Each Puzzlebot robot is equipped with holes and knobs on each side. The knobs have small hooks on them. When robots connect with each other, the knobs of each robot insert into the holes of the other robot, and the hooks keep the robot from decoupling through normal locomotion. Each Puzzlebot unit has two motors, a 3 V battery, and a WiFi Module, all handled by a custom Printed Circuit Board (PCB) onboard the robot. A figure taken from [31] can be seen in Figure 1.1 showing three PuzzleBots coupled together in order to cross a gap.

Each motor in the Puzzlebot robot is connected to a wheel. The robot runs in differential drive and is controlled using velocity commands received through the WiFi Module. The WiFi Module connects to a computer that sends velocities encoded as string. As of now, all computation is handled off-board the robot, due to the size

constraints of each module. After the implementation of sensors from this thesis work, the robot operates much in the same way. Data from the implemented sensors are sent via the WiFi module to the computer, so that computations can be done off-board before sending a string of the velocity commands back to the robot for execution.

1.3.2 Sensors



(a) (b) (c) (d)

Figure 1.2: (a) Swarm-bot, Mondada et al. 2006 [8, 28] (b) AMiR, Arvin et al. 2009 [1] (c) Kilobot, Rubenstein et al. 2014 [23] (d) e-puck, Mondada et al. 2009 [10]. These are examples of small robots that can be found in literature. These images are taken from their respective publications. All of these small robots utilize infrared sensors as one of their sensing modalities.

Other centimeter-scale robots in literature have a variety of applications, from existing as an educational resource to being able to explore real-world environments. Some examples of small robots can be seen in Figure 1.2. Due to the lack of available space on the robots, there are large size and power consumption constraints on the

sensors. Many sensors can be used for higher functionality, but at a cost for size. For example, each module in Swarm-bots boasts 50 sensors per unit, including multiple infrared sensors and a camera [8]. Not only is this very computationally expensive and requires more than the 3 V available to the PuzzleBots robots, but the robot is bulkier at 116 mm in diameter compared to PuzzleBots' 50 mm. It is also more difficult to construct than PuzzleBots. In a field setting, this is impractical as it will be hard to scale up and too costly to break or lose.

Existing implementations of small robotics use infrared sensors [1, 2, 9, 10, 12, 23, 26] and cameras [10]. Edge detection is commonly a task reserved for computer vision [34], but computer vision with a camera for gap detection would be computationally expensive, making them nonviable for PuzzleBots due to battery constraints and a limited microcontroller. Infrared sensors can detect ground clearance and distance.

Sensors can also be costly, but when navigating new environments, making expensive robots is not efficient, as they may reach areas where they cannot be retrieved. In the study of negative obstacle detection LIDAR is a possible choice for detecting gaps [25]. There exist off-the-shelf LIDAR sensors for about $25, but these sensors require 5 V and are single-point rangers. Thus, they are not likely to offer any more data than a 3 V ultrasonic sensor could offer but are much more expensive, since ultrasonic sensors can be bought for under $10.

1.3.3 Planning and Mapping

In addressing the difficulty of mapping a continuous space in a finite amount of memory, occupancy grids are a common solution, and work well in 2D [7, 17, 27]. Occupancy grids discretize the continuous space into smaller grid cells. Then, by using sensors to collect data about an environment, the robot can then mark each cell as free or occupied. The main benefit of an occupancy grid representation is its ability to handle sensor noise. Each occupancy cell can be represented as a probability by using a Bayes Filter on multiple sensor data readings to calculate the certainty of cell occupancy even in the presence of noise. This robustness to noise is what makes occupancy grids suitable even in cases where low-cost sensors are involved. For example, North et al. were able to use occupancy grids for Simultaneous Localization and Mapping (SLAM) with an ultrasonic sensor [18].

Focusing on the detection and avoidance of obstacles that come out of the ground has been studied mainly to avoid collisions [21]. However, there is much less research on detecting and avoiding gaps, also known as negative obstacles, in an environment. Existing methodologies include using LIDAR for gap detection and traversability analysis [25] and stereo-based vision for detection [13]. Both of these sensing methods are non-viable for Puzzlebots for the reasons described above.

Additionally, in Barua et al. [4], authors use an ultrasonic sensor to check for simple edge detection via ground clearance, but simply stop the robot when an edge is found. With Swarm-bot [8], Trianni et al. were able to use the onboard sensors for hole avoidance [28]. In order to avoid holes, each robot had a traction sensor and four infrared sensors. Each unit of Puzzlebots does not have the resources to support this sensor load. Beyond this, while the swarm-bot robots were able to avoid gaps, their aim was not to map the gaps, which offers a further challenge. The extra challenge of mapping the gap requires us to find a planning algorithm that allows the robot to traverse near enough to the gap where it is able to sense it, without falling off.

Following the edge of a gap would seem to be a similar problem as the well-studied line-following or wall-following problem. Line-following often relies on the use of multiple sensors [3, 4, 19] so that it can be determined which side of the robot the line is under. For example, in [3], at a 90° turn junction, if the line is detected underneath the right-sided sensor, then the robot will turn right. This kind of logic does not follow when trying to follow an edge on a platform gap, because the center of mass of the robot must always be on the platform, or else the robot will fall off. Similarly, some wall-following algorithm strategies like in [20] also rely on multiple sensors aimed specifically so that the robot can detect which side the wall is on and turn accordingly. The wall following problem has even been addressed in a robot with a single ultrasonic sensor [30]. However, the key assumption in this kind of wall-following work is that as you get closer or further from the obstacle, the sensor will still be able to sense it, but will read different values. This assumption does not apply when measuring a distance to a gap. Thus, a new kind of algorithm must be adapted for gap detection and following.

1.4 Contributions

With this thesis work, we aimed to move the PuzzleBots platform towards autonomy. The contributions of this work are laid out in this thesis as follows:

- Two different sensors are chosen for PuzzleBots in order to identify gap edges and gap size. The choices of these sensors were influenced by literature and constrained by the original system because the chosen sensors should minimally change the mechanical and electrical design of the original system.
- With these sensors chosen, we describe how they were characterized in order to understand what the best mounting position is for edge detection and mapping capabilities.
- Mechanical and electrical design changes to the original PuzzleBots system are proposed in order to support these new sensors.
- A simulation environment is provided, featuring a PuzzleBot robot equipped with sensors intended to emulate the functions of the hardware sensors. The simulation environment allows for the testing of new algorithms.
- Three different algorithms using the two sensors are described in order to show the capability of mapping a gap edge on a convex platform.
- The results of these algorithms are demonstrated in the simulation environment. We also show a basic safety stop on hardware to prove compatibility between the proposed solution and the original PuzzleBots system.

Chapter 2

Hardware Modifications

As discussed in the related works section of the Introduction, common choices for sensors for small robots are infrared sensors [1, 2, 9, 10, 12, 23, 26] or cameras [10]. Specifically for gap detection in robotics, sensors such as infrared [8], cameras [34], or LIDAR [25] can be used.

In this thesis, we aimed to integrate sensors on to the existing Puzzlebot platform with minimal hardware changes while keeping additional costs low. With this in mind, the sensors must follow the following constraints: (i) both sensors must be able to run under the same 3 V battery of the original system in addition to all the existing electronics, (ii) the sensors must use minimal computation, and (iii) the sensors must not increase the total cost of the robot significantly. Thus, cameras do not fit these constraints as they would have required too much computing power than we have available onboard. LIDAR, despite being highly accurate, is too expensive for this specific application.

Infrared sensors are capable of ground clearance detection and consist of two components: a transmitter and a receiver. For ground clearance, the infrared sensor can be angled towards the ground, and the transmitter sends out an infrared signal. If the receiver senses the return of the signal, then it can be determined that the infrared wave hit an object, and therefore there is no ground clearance. The receiver is from a class of sensors called photodetectors, which are sensors that detect light.

Infrared sensors would be a well-supported choice [1, 2, 9, 10, 12, 23, 26], as they can measure distance and can be used for edge detection with ground clearance

checking. In order to use them for PuzzleBots, two would be needed: one for ground clearance and one for distance detection, as the sensors for each use case would need to be mounted differently. However, they can be large, and including two on the robot would make the system significantly more bulky. Additionally, they typically require more than 3 V of power. For these reasons, infrared sensors do not fit our system constraints.

Despite the inability to use infrared sensors in their entirety, we are able to utilize just the photodetector component of the infrared sensor in order to detect ground clearance for the purposes of PuzzleBots. A common kind of photodetector is a photoresistor. A photoresistor can be purchased as a standalone sensor, is small, cheap, and requires minimal voltage to run. By using just the photoresistor component of the infrared sensor, we can still detect ground clearance without the high voltage requirements demanded by an infrared sensor. Therefore, we chose to use a photoresistor as the first sensor. The specific way a photoresistor can be used for ground clearance detection is described in the next section.

Photoresistors are low cost, small, and require very little voltage for power, and are thus an ideal choice for our application because they can be used to detect edges, but they cannot measure distance. Therefore, because we also aimed to give the robot the ability to detect the length of the gap, another sensor needed to be chosen. As a second sensor for distance detection, an ultrasonic sensor was chosen, because of its ability to detect distance and run under 3V. An additional benefit of using an ultrasonic sensor for distance instead of an infrared sensor, is that it is not light-dependent. Therefore, using the combination of a photoresistor and an ultrasonic sensor allows the system to be more robust to light changes, as opposed to using solely infrared sensors.

In order to determine the best use of these sensors for our application, before implementing the photoresistors and ultrasonic sensors onto the robot, experiments were run using Arduino Uno microcontrollers to characterize the sensors. All characterization experiments described in this chapter were run with sensors connected to an Arduino Uno, before they were integrated with the existing PCB. Data was collected by mounting the sensors to the robot, connecting them to an Arduino Uno, and moving the unpowered robot by hand.

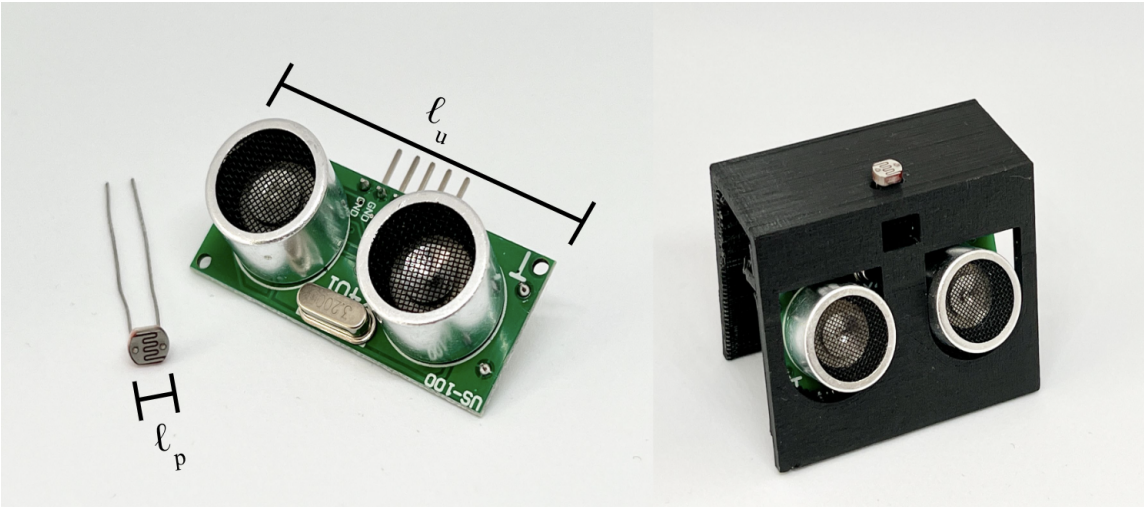


Figure 2.1: Photoresistor and Ultrasonic Sensors with $\ell_p = 0.5cm$ and $\ell_u = 4.4cm$ mount (left). The 3D printed mount used to house both sensors for Puzzlebot integration – the mount is upside down in this picture to show the photoresistor on the bottom (right). Different length mounts can be printed and interchanged as needed.

2.1 Photoresistors

Photoresistors were chosen due to their low voltage consumption and ease of use. A photoresistor is a variable resistor, where the amount of resistance the sensor provides is dependent on the amount of light it is reading – thus, they allow us to determine the amount of ambient light around the robot. The actual value reading of the photoresistor is not meaningful for our purposes, as the amount of light in the room could change from day to day, or the amount of voltage going through the resistor could fluctuate. However, mounting the photoresistor at the bottom of the robot with an angle towards the ground would allow the photoresistor to be exposed to more light when approaching an edge, since the platform would no longer block the sensor. Therefore, as a robot approaches an edge with a photoresistor, it can use the data from the sensor to detect large changes in light and interpret this as a platform edge.

In order to characterize this sensor, we were concerned with finding a mounting angle that would best suit the sensors for reliable data collection. We aimed to find sensor angles that would give a balance between the minimization of noise and

allow for the fastest detection of gaps. This section describes two sets of experiments run with the photoresistor. In both experiments, the chassis of the robot would be manually moved toward the edge of a table. The sensor would be connected to an Arduino, and the Arduino code would flag when it detected an edge, based on a light value threshold. In these experiments, the value of interest was the distance between the center of the robot and the edge of the table when the threshold flag was triggered. The threshold value for these experiments was found through calibration defined by averaging the photoresistor value over the table and over the gap at the time of the experiments.

The first experiment was run to gain a basic understanding of the capabilities of the photoresistor. In this experiment, the sensors were informally mounted onto the robot by taping them at the desired testing angles: 0°, 20°, 40° and 60°. The robot was slowly moved toward the edge of the table, as described above, and when the photoresistor value passed the threshold, the distance between the table and the front of the robot was measured. The distance from the center of the robot to the front of the robot is 2.5 cm – by adding 2.5 cm to the measured value, we can transpose the measured values to the distance between the edge and the center of the robot, which is the important value since the center of mass defines the configuration space.

From this experiment, it was determined that 0 degrees was not a viable mounting angle because it detected the edge only when the front of the robot had already passed the edge – because of this, 0° was removed from consideration. The 20° and 40° mounting angles had lower variation in distance values and were able to detect the edge before getting too close, while the 60° mounting had too much variation. Because of this, in the next round of experiments an additional 30° mounting angle was tested.

In this second set of experiments, the sensor was mounted at 0°, 20°, 40° and 60°, and was mounted by printing the bottom portion of the PuzzleBot chassis, with mounting holes that were accurately positioned to each of the desired testing angles. Again, the same distance data was collected by moving the chassis slowly toward the edge of a table.

From these experiments, it can be seen that 20° and 40° again performed better than the 60° angle, allowing the robot to detect an edge before the center of the robot moved too close to the edge of the table. However, with the added 30° mounting

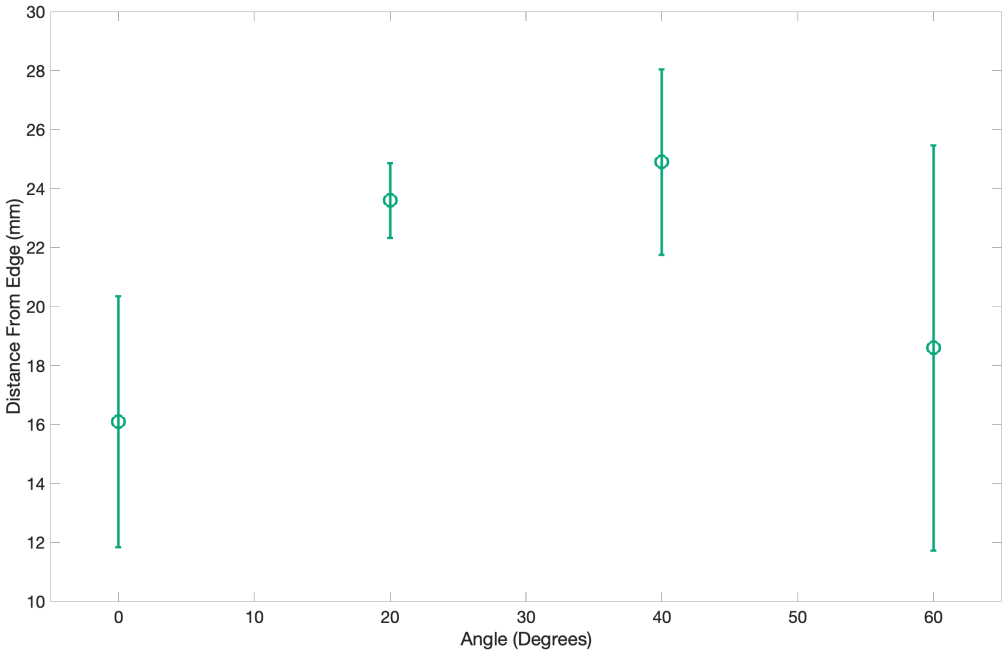


Figure 2.2: The average and standard deviation results of the first photoresistor experiments over ten trials for mounting angles of 0°, 20°, 40° and 60°. The distance value is the distance from the edge to the center of the robot when the gap was detected.

angle, similar distance values were obtained, but the standard deviation between distance values was lower. Therefore, from these second experiments, we determined that the ideal angle of the photoresistor is 30°.

2.2 Ultrasonic Sensor

Ultrasonic sensors were chosen because they are low cost, are able to detect distance, can be run under 3 V, and are unaffected by changes in lighting. They are also capable of detecting a large range of distances: from 2 cm to 450 cm, which allows for the detection of any gap within that distance range. In the case of Puzzlebots, this range addresses the problem scope because any gap smaller than 2 cm can be crossed without difficulty with a singular robot, and any gap larger than 450 cm would require 180 Puzzlebots to cross. In the following experiments, we focus on measuring gaps of size 0 cm to 12 cm – gap lengths that would require between zero robots to five

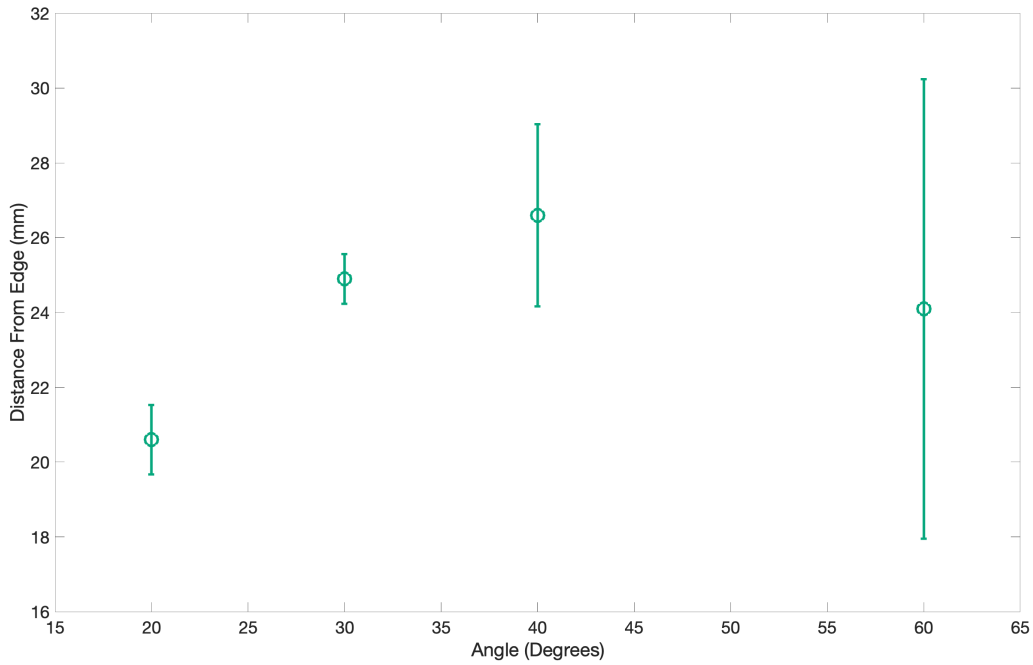


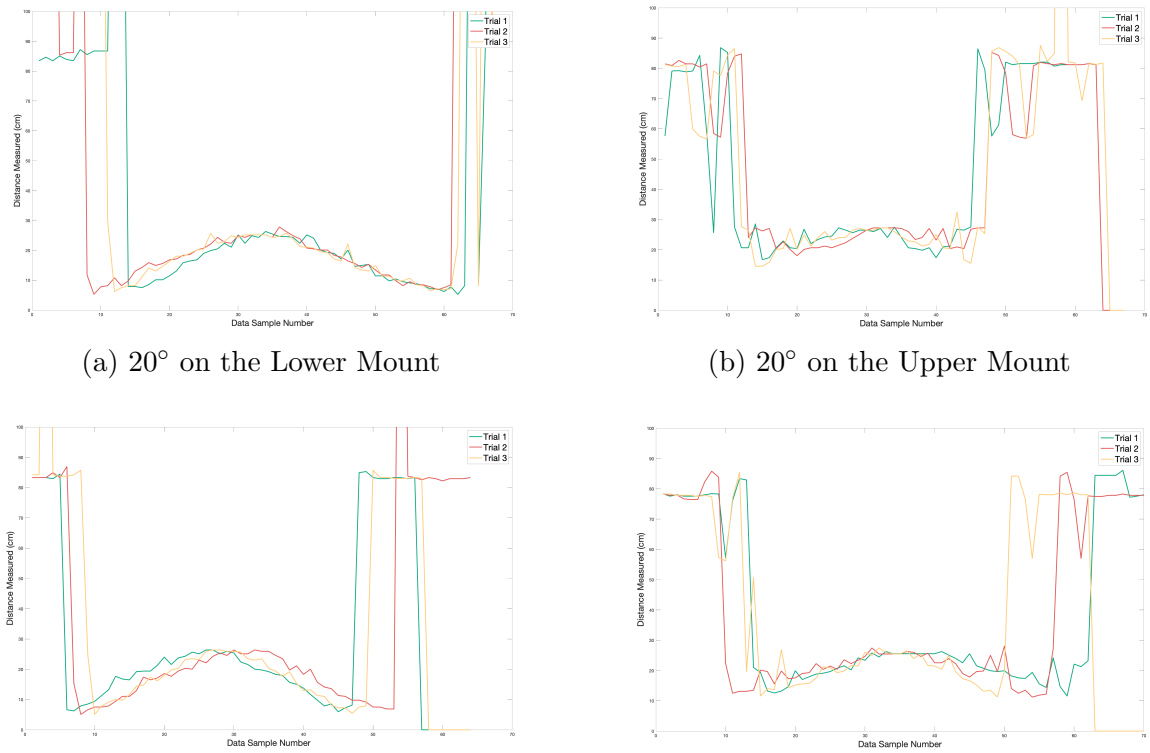
Figure 2.3: The average and standard deviation results of the second photoresistor experiments over ten trials for mounting angles of 0°, 20°, 40° and 60°. The distance value is the distance from the edge to the center of the robot when the gap was detected.

robots to cross.

In order to characterize the ultrasonic sensor, two main experiments were run. In the first experiment, "the Noise Experiments", the ultrasonic sensor was mounted at 15°, 20° and 25° from the vertical axis, towards the ground. During this experiment, the ultrasonic sensor was attached to the main chassis through a 3D-printed mounting piece that placed the sensor above the main body of the robot – we later denoted this kind of mounting as an upper mount. The goal of this experiment was to determine how the mounting angle affects the noise of the data readings when there is relative motion between the sensor and the sensed point.

The experiment was set up with the robot sitting at the edge of one platform, with a second platform flush with the edge. The ultrasonic sensor was mounted to the robot, and connected to an Arduino Uno that was feeding the stream of data to a laptop. During the experiment, the robot would stay in place while the second platform would be slowly moved away from the robot, and then moved again back

towards the robot, in order to create a heap shape in the data stream – in this experiment the distance values are not important, we are simply looking for the data stream with the least noise. For each angle, data from three trials were collected, and an example of the data collected can be seen in Figure 2.4. Because the box was moved by hand, the velocity of the box was not standardized between trials, and as such the slope of the profile may vary between trials, however, these experiments are still clearly able to differentiate the difference in noise level between mounts.



(a) 20° on the Lower Mount

(b) 20° on the Upper Mount

(c) 25° on the Lower Mount

(d) 25° on the Upper Mount

Figure 2.4: A sample of the results from the Noise Experiments showing two different angles on both the lower mounts (left) and upper mounts (right). Each trial of three is a single data stream from the ultrasonic sensor, with the sample number referring to the corresponding data point in the data stream. Some trials have been shifted slightly on the x-axis in order line up the data profiles. Because the upper mounts had significantly more noise, it was concluded that the lower mounts would be ideal.

In all three angles tested with this method, we found the data too noisy to be usable and expanded this experiment by adding a new mounting method that we denoted as a lower mount. In this mount, the ultrasonic sensor was mounted in front

of the robot, and at the same height. The difference between the upper and lower mounting systems can be seen in Figure 2.5. In addition to adding this lower mount, we added 10° as a testing angle, such that we tested 10°, 15°, 20° and 25° angles on the lower mount.

As can be seen in Figure 2.4, the results from the lower mount were significantly less noisy, allowing us to conclude that a lower mounting position would be best for implementation. However, from this experiment, we were unable to determine which angle would yield the most accurate distance reading and designed a second experiment to characterize this aspect of the ultrasonic sensor.

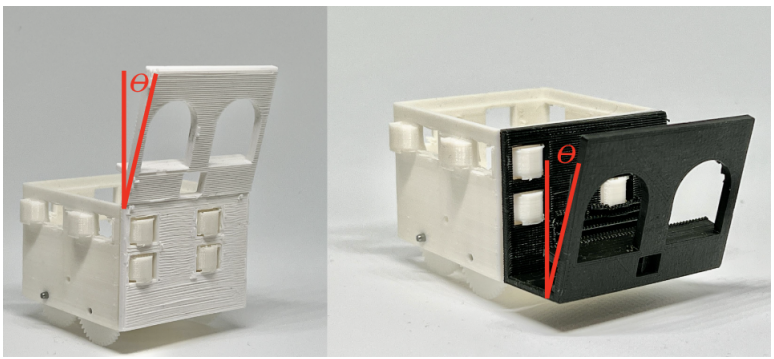


Figure 2.5: The upper mount is shown on the left. The lower mount is shown on the right. Shown here, θ is the angle of the mount changed during these experiments. In this figure, $\theta = 15°$

.

The second set of experiments involved the sweep of three different parameters: the angle of the mount, the distance between the robot to the edge of the platform it is on, and the distance of the gap. The goal of this experiment was to determine the best orientation of the ultrasonic sensor in order to yield accurate values for different gap sizes. All data collection was performed using a lower mounting system since the previous experiment had shown this to be best.

Table 2.1: Distance parameters used for the second ultrasonic experiment. All values are in centimeters.

d values	x values
0	0
2	3
4	6
6	9
8	
10	
12	

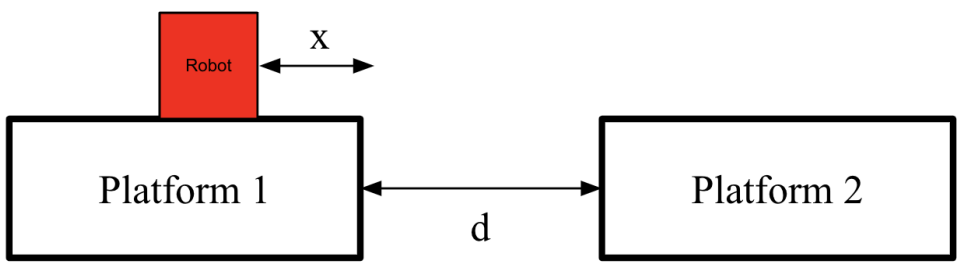


Figure 2.6: A schematic showing the setup of parameters for the second ultrasonic experiments. The values of d and x used during this experiment can be seen in Table 2.1. During each run of the experiment, the distance value of x would be set and Platform 2 would be moved by hand in order to vary d.

In each trial during this second experiment, one of four sensor angles was tested (10°, 15°, 20° and 25°), and the robot was placed at a distance x away from the edge of a gap. A second platform was placed, flush to the edge of the platform the robot is on. At this point of the experiment, the variable d in Figure 2.6 is 0 cm. The sensor is then turned on, and data collection starts. During data collection, platform 2 is dragged further away from platform 1. It is stopped for 5 s at a time when d is changed to each of the values in Table 2.1. A schematic for the setup of this experiment can be seen in Figure 2.6.

At each of the 5 s intervals when platform 2 is a distance d away from platform 1,

the data collected from the ultrasonic sensor should plateau. The evaluation criteria for which angle mounting was best, was the deviation between distance measured by the sensor at each plateau against the ground truth distance value determined by x and d. An example of the distance data collected for the 10° mount can be seen in Figure 2.7.

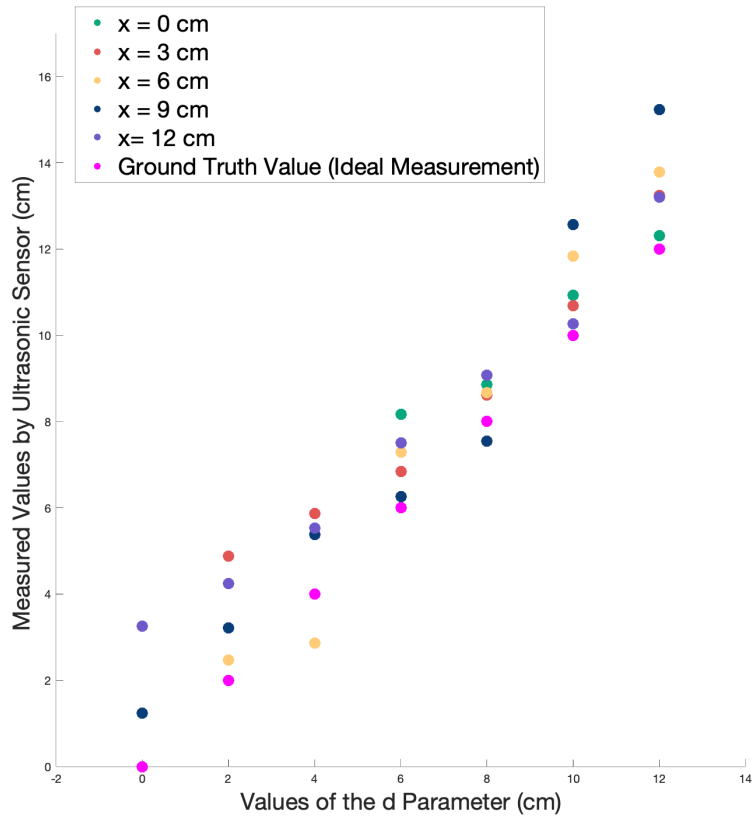


Figure 2.7: The post-processed data that was collected with the 10° lower mount for the second experimental setup. The distance measured values shown here represent the measured horizontal distance of the gap, after processing the raw sensor data values to account for the sensor angle and the x value of the robot location. In other words, it is the d value that the robot would use as the gap length. Each value seen in this figure is one point taken from each plateau that is seen in the raw data, which can be viewed in Appendix Figure A.1a. Not all of the x values have seven corresponding points for each d parameter – this is due to the data being too noisy to extract a value at these values, so they are disregarded.

Due to the ultrasonic sensor being mounted at a certain height (h = 3 cm, in the

case of the lower mount), the sensor should have a different minimum gap distance that it would be able to sense at each mounting angle since the sound wave needs to vertically travel more than the distance from the sensor to the ground in order to reflect off the side of a new platform. In theory, if there is not enough horizontal distance for the sound wave to travel in order for this height to be lost, then it would just sense a distance of

$$\frac{h}{tan(\alpha)} \tag{2.1}$$

to the top of the next platform, where h is the height of the sensor and α is the angle of the sensor as shown in Figure 2.5. For example, for a 10 degree mount, the lowest value that the sensor should be able to detect is 17 cm, as anything lower than that, the sound wave would pass over the entirety of the gap. However, as Figure 2.7 shows, this is not what happens in reality. This can likely be attributed to the sensor's measuring angle – while ideally, the sensor measures an object that is in line with a perpendicular ray originating from the sensor face, the sensor's sound wave is able to detect objects within a certain angle range of a perpendicular ray.

The results of the other angles can be seen in the Appendix. In these experiments, it was determined that the 10° mount would be the most ideal orientation. Though the 15 degree mount values seen in Figure A.2b are more accurate to the ground truth values, the 10° mount values have less variation between measured values and the ground truth values over varying parameters. The accuracy of the values is of less importance in this case, because they can be shifted towards the ground truth, or if using the unmodified values for mapping, as seen in Figure 2.7 the measured values are generally higher than the ground truth values – which would yield a higher factor of safety when trying to cross a gap. Furthermore, as can be seen in the raw data in A.1, the data from the 10° mount is much less noisy than the rest of the data, as can be seen, because the plateau values are much more distinguishable in the 10° data than in the other angles.

2.3 Integration

During very initial experiments, the sensors were taped on at an angle before any mounting elements were 3D-printed. This was just for very initial testing, to get

a general sense of how the sensors should be mounted. Then, a more permanent solution needed to be created for integration.

During photoresistor testing, we integrated the sensors into the main robot chassis, so that the main body of the robot, including the sensor mount, would all be 3D printed as one piece. While testing this original concept just by printing the bottom half of the chassis with the mounting holes, it was discovered that the prints of the full chassis take a long time. Even printing just the bottom with the mounting holes was a multiple-hour-long print. Ultimately, when making many robots, this is not feasible for the long term, as it makes any existing robot chassis obsolete.

Instead of printing the mount directly on the chassis, we created a separate piece mounting system as seen in Figure 2.8. Because the Puzzlebots are already designed to connect to each other, we used these same connection points on the robot as mounting holes, so that mounts can be printed as needed for existing robot chassis without any additional alterations of the original system. A single mount also holds both the ultrasonic and photoresistor sensors.

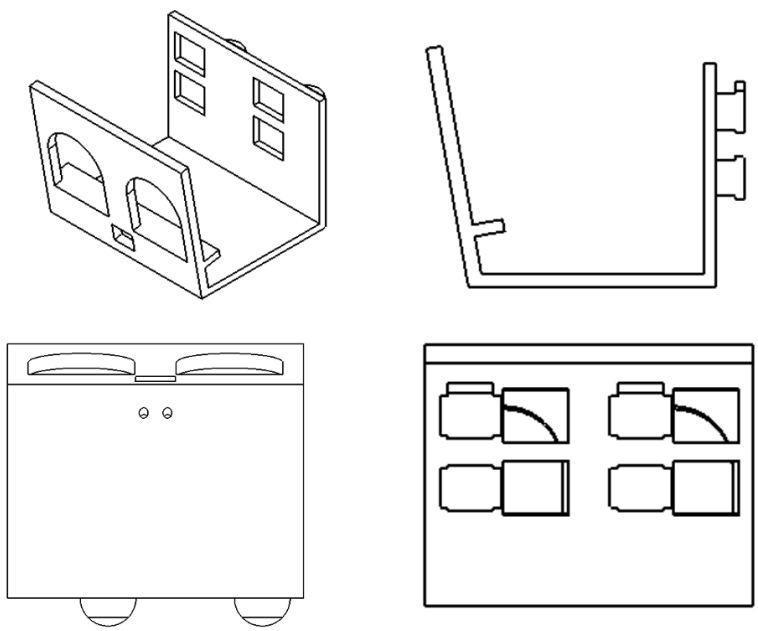


Figure 2.8: Isometric, side, bottom, and top view of sensor mount drawings.

The mount connects to the front of the robot. This also gives the additional

benefit of having the sensors far in front of the center of mass of the robot, giving the sensors a greater chance of detecting a gap before the center of mass nears the edge of the configuration space.

With regards to electronic changes made to the robot, because sensors were chosen with the specific goal in mind of keeping the main body of the robot the same, the robot is still able to run with the same 3 V battery. The PCB was minimally modified to include the additional sensor wiring points as seen in Figure 2.9.

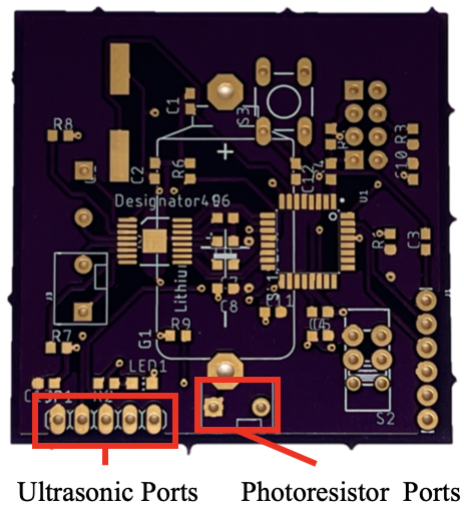


Figure 2.9: The updated PCB to include sensors for Puzzlebot.

2.4 Bill of Materials

While during design, there was no hard maximum on the cost of each Puzzlebot unit, the overall goal when choosing parts was to keep the cost low. The following are the parts included in the robot integration including the components from the original Puzzlebots paper [31].

Table 2.2: Bill of Materials Table

Part Name	Cost Per Unit	Quantity	Total
PCB	$5.68	1	$5.68
Chassis*	$3	1	$3
CR2 Battery	$2.45	1	$2.45
DC Motor	$3.64	2	$7.28
ESP8266 WiFi Module	$6.95	1	$6.95
Gears and Rods	$1.60	1	$1.60
Ball Bearings	$2	2	$4.00
Small Electronics**	$7.21	1	$7.21
Photoresistor	$0.20	1	$0.20
Ultrasonic Sensor	$6.95	1	$6.95
Sensor Mount***	$0.50	1	$0.50
Final Total			$45.82

*Cost of 3D Printed TPU Mass

**Small Electronics such as Surface Mount Resistors, Capacitors, Wires, etc.

***Cost of 3D Printed PLA Mass. The material of the mount is not important. It can be printed in any rigid 3D printable material.

General Note: The price for smaller components in the table above is for when the components are bought in bulk [31].

The addition of the sensors onto the robot only increases the cost of a Puzzlebot unit by $7.65, keeping the overall cost of one Puzzlebot unit equipped with sensors to $45.82. Because detecting gaps is an inherently unsafe task, due to the need for the robot to venture near the edge in order to sense the gap, it is imperative that the cost of one robot is low. Because the cost is low, it allows the robot to prioritize exploration over safety and allows the robot to map edges with the sole goal of providing as much environmental information as possible to future robots without excessive concern for their own safety.

Ultimately, the choice of these sensors for use for edge detection is beneficial beyond the scope of Puzzlebots because they can be easily implemented onto any existing mobile robotic system with minimal additional cost. They are also easily available and have many supporting resources easily found online, such as the code

for collecting the sensor data. Finally, as shown in the implementation section of this chapter, they can be implemented into the system with minimal hardware changes.

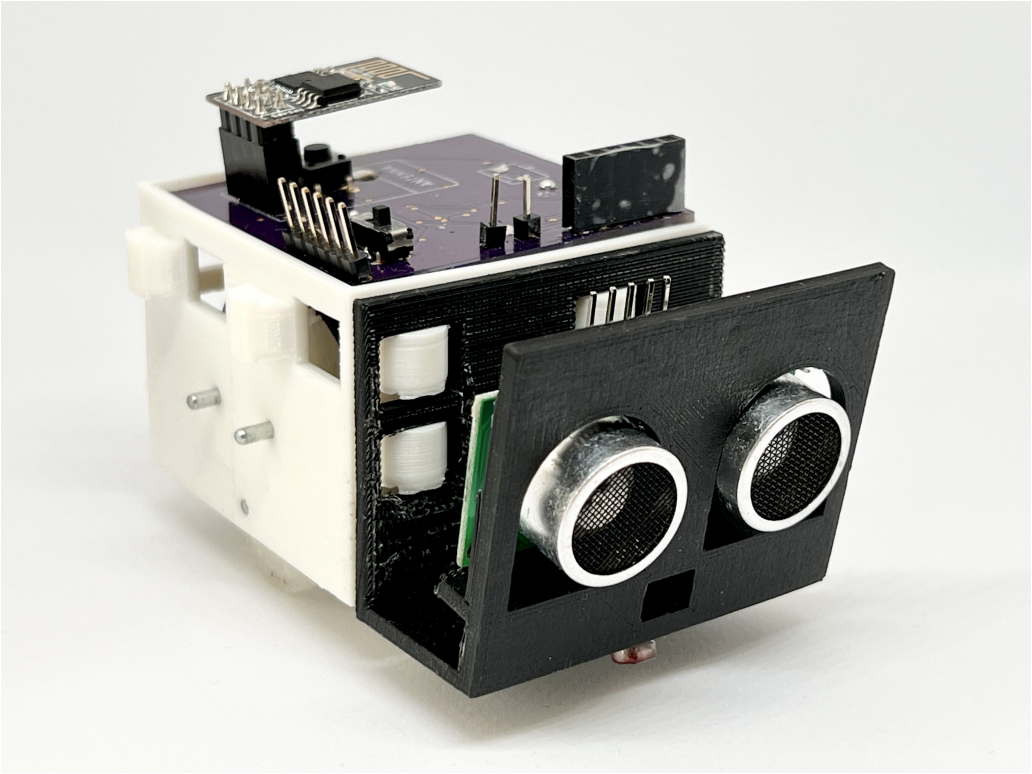


Figure 2.10: The assembled robot with mounted sensors.

Chapter 3

Simulation Environment and Algorithms

Before using the fully integrated system of sensors on hardware, we implemented a simulation environment in CoppeliaSim, whose website can be found at the following hyperlink: CoppeliaSim Website. In the simulation, there is one Puzzlebot equipped with two different sensors, meant to mimic the data of the real-life ultrasonic and photoresistor sensors. The simulation was used for the development and testing of three different algorithms.

3.1 Setup

In each simulation environment used to test the algorithms, there is the robot equipped with two sensors, a turquoise platform, and a black platform. The robot sits on a turquoise platform while the black platform represents the other side of the gap that the robot is trying to map. The black platform is a simple rectangle in all simulation environments used, and the robot never physically interacts with it – it is only needed to show the ability of the ultrasonic sensor to map the length of the gap. On the other hand, the turquoise platform is varied between simulation environments. Three shapes of the turquoise platform were used to test the algorithms presented in this chapter: a circle to demonstrate their capabilities on a basic curved edge, a hexagon to demonstrate their capabilities on a straight edge, and an oval to demonstrate their

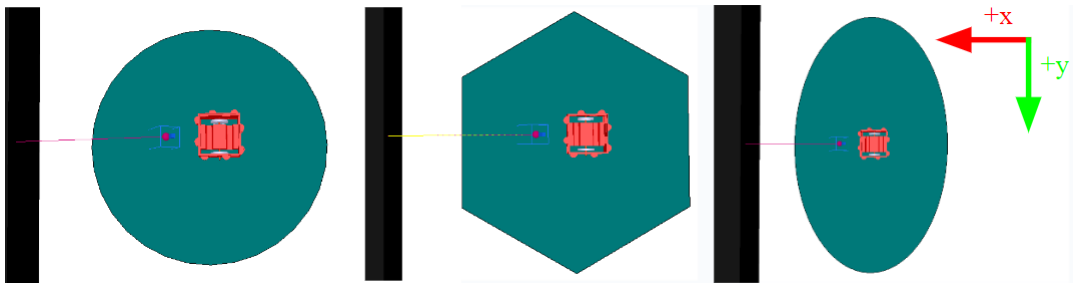


Figure 3.1: Three different CoppeliaSim Simulation environments used for algorithm testing. The robot traverses and maps the turquoise platform. There is a black platform to the left of each platform that the robot, shown in red, senses and uses to measure the distance of the gap with the ultrasonic sensor.

capabilities on an asymmetric shape. The setup of these environments can be seen in 3.1.

3.2 Sensors in Simulation

CoppeliaSim does not offer a sensor model to measure ambient light that acts in the same way as the photoresistor. As a result, a vision sensor with a perspective view was used in the simulation to emulate the functionality of the photoresistor – the vision sensor acts similarly to a camera by giving pixel RGB values of what it can see. The robot sits on a turquoise platform with RGB values of [0, 0.5, 0.5], while the floor beneath has RGB values of [0, 0, 0]. In the simulation, we emulate the measurement of ambient light by averaging the pixel values of the red channel – as the robot approaches the gap, the vision sensor will pick up more white pixels and the average value of the red channel will decrease. Thus, the functionality of the vision sensor can be used to emulate the photoresistor.

The implementation of the ultrasonic sensor in the simulation was much simpler – CoppeliaSim has a built-in single-point "Proximity Sensor" that measures the distance to the nearest point in a ray originating from the sensor. Thus, it behaves the same as an ultrasonic sensor.

In the hardware chapter of this thesis, we presented experiments as a rationale for the choice of mounting orientation. These mounting choices were replicated in the simulation in order to best represent the hardware: the vision sensor was angled to

point 30° from the floor away from the robot while the proximity sensor was angled to point 10° towards the floor from horizontal. The sensors are also positioned to be in front of the robot in simulation, approximately the same distance away from the center of the robot as they are in hardware. This distance can also be changed easily as needed in both simulation and hardware to improve results, as needed.

For the rest of this thesis, when discussing these simulated sensors, we refer to the vision sensor as a photoresistor and the proximity sensor as an ultrasonic sensor in order to maintain consistency with the hardware discussion.

3.3 Algorithms

We now present the mapping methodology used and three planning algorithms of increasing complexity to demonstrate the ability of the robot to identify a gap and do a partial or full mapping of the gap. Algorithms were developed in Python. The following assumptions were made in the development of these algorithms:

- The platform has a single continuous edge: there are no gaps in the middle of the platform. Because of this, the algorithms favor mapping the entirety of the platform edge as quickly as possible, foregoing a frontier-based exploration approach [29].
- The platform must be convex. This assumption is particularly important for the third algorithm presented in this chapter, which relies on boundaries created by half-planes.
- The platform must be level. This is because the mapping of safe, traversable areas is based on a lack of height change as will be described in the following mapping subsection.
- The platform on the other side of the gap must either be at the same height or lower than the current robot's platform. This assumption is necessary because, despite the fact the distance sensor will properly sense the wall of the platform, the robot will not be able to traverse the gap if it is higher than itself.
- The platform must not have additional obstacles other than the gap. Implementing sensors that detect other obstacles besides the gap is out of the scope of this thesis.

- The length of the gap must not be more than 30 cm. This assumption is important because value readings higher than 30 cm from the distance sensor are ignored in order to distinguish between sensing another platform versus sensing the floor below the platform. Additionally, this assumption will be necessary for hardware as noisy values from the ultrasonic sensor are high values, as seen from the experiments in the hardware chapter of this thesis. This is an assumption within the scope of this thesis because a gap of 30 cm is six times the length of one Puzzlebot robot and would require at least 12 robots to cross the gap.

Each algorithm has a set of hyperparameters that are described in Table 3.1.

3.3.1 Mapping

The same mapping procedure is used for all three of the algorithms and is described once here in this subsection, but omitted from the following algorithm descriptions and pseudocode for conciseness.

In order to create a map of the environment, we use occupancy grids[7, 17, 27] in order to designate positions as traversable or as a gap. The occupancy grid is stored as a 2D Numpy array of size N x N such that N is an odd value, and each cell in the occupancy map represents a 1 cm $\times$ 1 cm area of the environment. The starting pose of the robot is stored and used as the origin during each simulation run to ensure that the starting location of the robot does not affect the map output or algorithm. Specifically, we can represent the starting pose of the robot as $[x_s, y_s]$ in the world frame. Then, once the simulation starts and the world pose is known, $[x_{world}, y_{world}]$ is translated to the relative frame defined by the starting position such that $[x_t, y_t] = [x_{world}, y_{world}] - [x_s, y_s]$ and this relative position is used for mapping. Therefore, the starting position of the robot will always have an index [N//2, N//2] in the occupancy grid, where // denotes integer division.

In addition to this occupancy grid, there is also a corresponding uncertainty matrix that is the same size as the occupancy grid and uses the same indices. The uncertainty matrix represents how certain the robot is of each occupancy cell reading – with the freespace assumption this uncertainty matrix is initiated to be composed entirely with 1. Because this is in simulation and the sensor values are absolute

readings without noise, when a cell is sensed to be a gap, this uncertainty goes to 0 in the uncertainty matrix. Similarly, when the robot traverses a cell without suffering a significant height differential from the robot starting point, this also sets the uncertainty value at that position to zero.

The map is initiated according to the free space assumption, such that all cells are marked as traversable until sensed otherwise. When a robot senses a gap with the photoresistor, it marks the sensed cell as a gap. When this occurs, the robot also reads the distance value measured from the ultrasonic sensor and maps all the cells in a line from the sensor location to the sensed distance point as a gap.

3.3.2 Basic Safety Algorithm

The most basic of the three algorithms, this first approach is a simple demonstration of the basic functionality of the sensors and it may be used to prevent a robot from falling off an edge. In this algorithm, the robot is given a forward velocity and sustains this forward motion until a gap is detected. A gap is detected when the vision sensor reading as described above, goes beyond a set threshold that is a hyperparameter of the algorithm. When the gap is detected, the robot stops all motion and thus ends the algorithm.

Algorithm 1 Basic Safety Algorithm
With hyperparameters forward_velocity and light_threshold from Table 3.1

```
while Running Algorithm do
    LightVal ← LightSensorReading()
    if LightVal > light_threshold then
        Stop
    else
        [v, w] ← forward_velocity, 0
        Send v, w to robot
    end if
end while
```

3.3.3 Parking Algorithm

Algorithm 2 Parking Algorithm
With hyperparameters forward_velocity, light_threshold, backward_velocity, and angular_velocity from Table 3.1

```
while Running Algorithm do
    LightVal ← LightSensorReading()
    if LightVal > light_threshold then
        while LightSensorReading > light_threshold do
            [v, w] ← backward_velocity, angular_velocity
            Send v, w to robot
        end while
    else
        [v, w] ← forward_velocity, 0
        Send v, w to robot
    end if
end while
```

This algorithm extends the basic safety algorithm by adding the goal of following the edge of the platform. In trying to follow the edge of the platform, the most intuitive motion may seem to be to go straight and only turn when an edge is detected until the edge is out of sight. However, this is flawed, because the sensor is only able to give a binary reading between edge versus no edge, which means that the edge will still be nearby after the turn has been completed. However, in this case, each time the robot continues going straight, it slowly continues getting closer to the edge until it has fallen off. To counteract this occurrence, when the robot detects that the vision sensor is reading above the light threshold hyperparameter in this second algorithm, it both backs and turns away from the edge until the edge is no longer detected. The motion for this backward and turn away motion can be seen in Figure 3.2 and resembles the motion of a car pulling out of a head-in a parking space. By adding this backward motion to pull away from the gap, the robot is able to detect different points along the edge of the platform without immediately getting too close and falling off. The exact linear and angular velocity that the robot uses during this motion is a hyperparameter and can be seen in Figure 3.1. Note that the hyperparameters for this algorithm are a strict superset of the hyperparameters of the basic safety algorithm.

While this algorithm succeeds at mapping the edge of the platform as will be

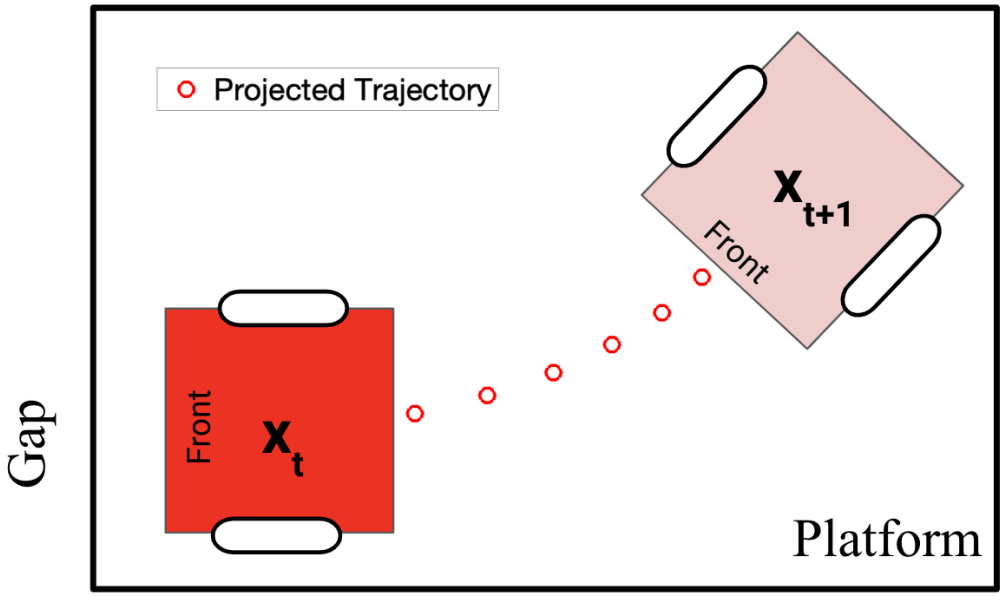


Figure 3.2: An example trajectory of the robot moving away from the edge for the Parking Algorithm. When the solid red robot detects it is at the edge of the platform, it pulls away and moves to the location of the faded red robot, allowing it to move forward to another point along the edge without falling off.

shown in the next chapter, it does not use any of the learned knowledge of the platform to inform its motion planning decisions. As a result, while it is successful at providing future robots with information about the platform and offers a layer of safety in future robot traversals, it does not benefit from any safety constraints.

3.3.4 Linear Constraint Algorithm

In this algorithm, we build upon the first two algorithms further – we maintain the parking motion when a gap has been detected, and add onto this with the creation of linear constraints based on sensed points. The goal of this algorithm is to create a half-plane from each pair of most recently sensed edge points – to draw a line that intersects these points and constrain the robot to be within the half-plane that contains the platform. The safe region, therefore, would be the region of intersection of all created half-planes. An ideal set of constraints for the hexagonal platform can be seen in Figure 3.3.

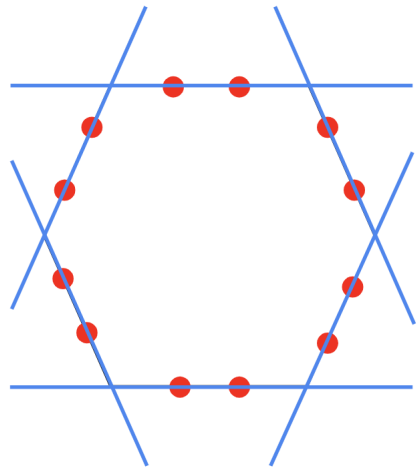


Figure 3.3: A possible ideal set of constraints for a hexagonal platform, where the red dots represent the sensed points along the edge of the platform and the blue lines represent the linear constraints.

The creation of the linear constraints is based on the center of the occupancy grid cell positions. For instance, all sensor poses with relative x positions of [0, 1] will use an x position of 0.5 to create the linear constraint. This methodology is used to prevent very angled linear constraints from being created in the event of two points being sensed very close to each other.

In this algorithm, the robot starts by going straight as it also does in Algorithm 1. Once an edge is detected, it does the parking motion from Algorithm 2 and also saves this sensed point. It then goes forward again to detect a second edge point and creates a linear constraint between these two points. Once at least one linear constraint has been created, the algorithm transitions from automatically going straight when an edge is not detected to attempting to calculate a linear and angular velocity pair that will keep the robot inside all of the defined half-planes for the next time step (t+1) state.

The minimization problem is formulated as follows: given an optimal policy velocity (the hyperparameter u^*), each linear constraint created from the sensed points is put into the form of one of the following:

$$A_{lb}x_t \geq b_{lb} \tag{3.1}$$

$$A_{ub}x_t \leq b_{ub} \tag{3.2}$$

where x_t is the current state of the robot. The goal of the minimization is to find a velocity that is closest to the optimal policy such that the state of the robot in the next time step, x_{t+1} adheres to

$$A_{lb}x_{t+1} \geq b_{lb} \tag{3.3}$$

$$A_{ub}x_{t+1} \leq b_{ub} \tag{3.4}$$

Where A_{lb} and b_{lb} are matrices corresponding to the lower bound position limits and are of size Nx2 and Nx1 respectively with N as the number of lower bound constraints. Similarly, A_{ub} and b_{ub} are matrices corresponding to the upper bound position limits and are of size Mx2 and Mx1 respectively with M as the number of upper bound constraints. Only the x and y positions of the state of the robot are used in this minimization as there are no constraints on the orientation of the robot. Therefore, x_t and x_{t+1} are both matrices of size 2x1.

Since half-planes pose constraints on position while the minimization returns a velocity u, we must define how the future state x_{t+1} is a function of u. We start with the Jacobian [32] that defines the motion of a point d centimeter in front of the center of the robot along the x-axis:

$$J = \begin{bmatrix} cos(\theta_t) & -dsin(\theta_t) \\ sin(\theta_t) & dcos(\theta_t) \end{bmatrix} \tag{3.5}$$

from the transition equation [14] that defines x_{t+1} for differential drive motion, where θ_t is the orientation of the robot at the current timestep. We can therefore formalize the minimization problem by using a discretization [32] of the transition equation,

leaving us with:

$$
\begin{aligned}
\min_{u} \quad & ||u - u^*||^2 \\
\text{s.t.} \quad & x_{t+1} = x_t + Ju_t dt \\
& A_{ub} x_{t+1} \leq b_{ub} \\
& A_{lb} x_{t+1} \geq b_{lb} \\
& [v, w] = u \\
& v \in velocity\_bounds \\
& w \in angular\_velocity\_bounds
\end{aligned}
\tag{3.6}
$$

The algorithm solves this minimization problem using the Python module Scipy – to which we feed in hyperparameters of optimization, optimal velocity, and velocity bounds. The full list of hyperparameters for this Linear Constraints algorithm can be seen in Table 3.1 and again are a superset of the hyperparameters of the Parking Algorithm discussed above.

3.3.4.1 Challenges with the Linear Constraint Algorithm

Two prime challenges worth noting arise with this algorithm:

1. The linear constraints are created from the two most recently sensed points, which can create lines that cut directly through the valid areas of the platform for two reasons:
    - The data from the vision sensor is only capable of relaying a binary data stream regarding if the sensed point is a platform versus not a platform. Therefore, if two points in the gap are sensed immediately after one another while the robot moves straight, this will create a linear constraint that goes through the center of the robot, and through the platform.
    - Based on the linear and angular velocity sent to the robot, the robot may skip large sections of mapping an edge and as a consequence will sequentially sense two points far away from each other on different edges of the platform. Then, when the algorithm tries to create a line between

these two points, the line will cut through a large section of the valid platform.

2. Despite the linear optimization that happens in each loop of this algorithm, the optimizer may be unable to find a linear and angular velocity pair that allows the robot to reach a safe area in the next time step. In this scenario, the optimizer tries to minimize the objective function and violates the constraints. Therefore, the linear constraints do not offer any safety guarantees.

In order to mitigate the effects of the first challenge, we implemented four different methods of filtering out linear constraints:

1. For this first method, we introduce the "okFlag" as seen in the pseudocode. With the okFlag, we only use a sensed gap point to create a linear constraint if the previous sensed point was not a gap. In other words, if we assign points sensed over a gap a value of 0 and points sensed over a platform a value of 1, then only the points where the robot crossed the threshold of reading 1 to 0 will be used to create linear constraints.
2. In order to facilitate this second method, in each loop of the algorithm, the robot stores its pose in a list designated as visited points. In order for a linear constraint to be created, all of the previously visited points must adhere to the newly created linear constraint or the linear constraint is disregarded.
3. As a third method, we also only use points in the linear constraints that are newly mapped. That is, in order for a point to be used to create a half-plane, it must have turned from a value of 1 to 0 in the occupancy grid in the same loop during which it was sensed.
4. Finally, in order to prevent points far away from each other from creating a linear constraint, we only consider linear constraints from two points that are within a set Euclidean distance away from each other.

Note that all four of these methods are implemented into this algorithm, but only the okFlag is shown in the pseudocode in order to keep the pseudocode shorter and easier to understand. Also, note that while these four methods greatly lower the frequency of generating problematic constraints, some constraints that cut through portions of the platform may still be created.

Because this second challenge is more difficult to address since the nature of trying

to detect a gap is unsafe to work due to the robot's need to be near enough to the edge of the platform in order to sense it, we instead recognize that this algorithm does not offer any safety guarantees. However, we present this algorithm because it offers an additional level of safety compared to the Parking Algorithm – when the optimizer is able to find a velocity that satisfies all the constraints, the motion of the robot will be known to be safer in these timesteps. Introducing the linear constraints of this algorithm also provides additional flexibility in controlling the motions of the robot – through the ability to change its many hyperparameters – that is not possible in either the Basic Safety Algorithm or the Parking Algorithm.

A contributing factor to this challenge is the non-holonomic constraints of the Puzzlebots robot – namely its inability to move side to side instantaneously. Though out of the scope of this thesis, a method of mitigating this second challenge is to allow the robot to move side to side instantaneously, which would give the robot an additional instantaneous degree of freedom through which it may be able to leave from or avoid going into an unsafe region.

Algorithm 3 Map Boundary with Linear Optimization
With hyperparameters light_threshold, optimal_velocity, optimal_angular_velocity, forward_velocity, angular_velocity, backward_velocity, and buffer as described in Table 3.1

```
okFlag ← True
Constraints ← ∅
x1 ← None, y1 ← None, x2 ← None, y2 ← None
u* ← [v*, w*]^T
while Running Algorithm do
    LightVal ← LightSensorReading()
    DistVal ← UltrasonicSensorReading()
    PosSensorX , PosSensorY ← PositionReading() + buffer
    if LightVal > light_threshold then
        if okFlag is True[1] then
            x1 ← x2
            y1 ← y2
            x2 ← PosSensorX
            y2 ← PosSensorY
            if x1 ≠ None then
                newLine ← Create a line of the form Ax=b s.t.
                              x = [posSensorX, posSensorY]^T
                newConstraint ← Constraint from newLine following the form
                                of either Equation (3.1) or (3.2) s.t x_t in
                                Equations (3.1) and (3.2) is set to [0, 0]^T
                Constraints.insert(newConstraint)
            end if
        end if
        while LightSensorReading > light_threshold do
            [v, w] ← backward_velocity, angular_velocity
            Send v, w to robot
        end while
        okFlag ← False
    else
        okFlag ← True
    end if
    if Constraints ≠ ∅ then
        [v, w] ← Solve minimization of u = [v, w]^T according to Equation (3.6)
                 with Constraints.
        Send v, w to robot
    else
        [v, w] ← forward_velocity, 0
        Send v, w to robot
    end if
end while
```

[**1**] The okFlag is one of four methods used to filter out Linear Constraints – the other two are not shown here in this pseudocode because it would overcomplicate the pseudocode without adding understanding to the core of the algorithm. For explanations on all four methods, see challenge 1 in the above subsection.

Table 3.1: Hyperparameters of Algorithms with Descriptions. Let the set of hyperparameters of the Basic Safety, Parking, and Linear Constraints Algorithms to be represented by $H_b, H_p,$ and H_l respectively then $H_b \subset H_p \subset H_l$

Algorithms	Hyperparameter	Description
Basic Safety	light_threshold	The ambient light threshold used as the gap detection condition.
	forward_velocity (v)	Forward velocity that is sent to the robot before the gap is detected.
Parking	light_threshold	The ambient light threshold used as the gap detection condition.
	forward_velocity (v)	Forward velocity that is sent to the robot when the gap is not detected.
	angular_velocity (w)	Angular velocity sent to the robot when the gap is detected.
	backward_velocity	Backwards velocity sent to the robot when the gap is detected.
	buffer	The distance away from the center of the robot that is mapped as an edge.
Linear Constraints	light_threshold	The ambient light threshold used as the gap detection condition.
	forward_velocity (v)	Forward velocity that is sent to the robot when the gap is not detected and when there are no linear constraints.
	angular_velocity (w)	Angular velocity sent to the robot when the gap is detected.
	backward_velocity	Backwards velocity sent to the robot when the gap is detected.
	buffer	The distance away from the center of the robot that is mapped as an edge.
	optimal_velocity (v*)	The optimal forward velocity that the optimizer tries to match.
	optimal_angular_velocity (w*)	The optimal angular velocity that the optimizer tries to match.
	velocity_guess	Parameter fed into the scipy.optimize.minimize function that determines where the optimizer starts looking for forward velocities.
	angular_velocity_guess	Parameter fed into the scipy.optimize.minimize function that determines where the optimizer starts looking for angular velocities.
	velocity_bounds	Set of lower and upper bounds on the velocity return for the minimizer.
	angular_velocity_bounds	Set of lower and upper bounds on the angular velocity return for the minimizer.
	dt	Length of one time step. Affects the prediction for where the robot will be in next time step.
	optimization_method	Parameter fed into the scipy.optimize.minimize function call that determines the method of minimizing the objective function

Chapter 4

Demonstrations

4.1 Simulation Demonstrations

In this section, we present the maps created as a result of running the three algorithms in the CoppeliaSim Environments.

4.1.1 Basic Safety Algorithm Results

In order to test hte efficacy of the basic stop algorithm, we run the algorithm five times on each of the platform shapes shown in Figure 3.1, and report the success rate. During each run, success is determined by whether or not the robot successfully stops before falling off the edge of the platform.

Table 4.1: Success rate on each of the three platform shapes over five trials for the basic safety algorithm

	Circle	Hexagon	Oval
Success Rate	100%	100%	100%

Despite the simple algorithm, this method is imperative in showing the basic abilities of the implemented sensors. It demonstrates the safety added by having the sensors on board the robot.

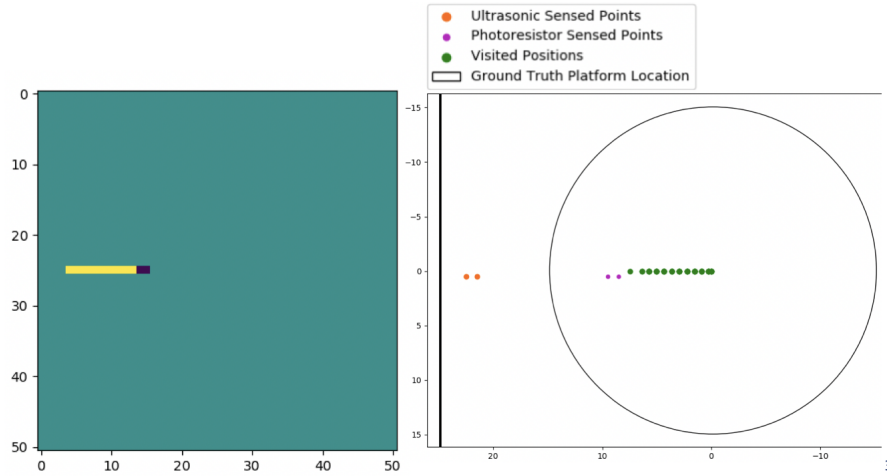


Figure 4.1: The occupancy map is shown on the left, where teal represents free space, yellow represents gaps sensed via the ultrasonic sensor, and purple represents gaps sensed via photoresistor. There is a straight yellow line due to the filling of the occupancy map with gap values between the sensed first platform edge all the way to where the ultrasonic sensor has sensed a second platform. On the right, the graph shows the visited points, the sensed ultrasonic points, and the sensed photoresistor points.

4.1.2 Parking Algorithm Results

In this section, the gap maps from the Parking algorithm are presented based on their performance in simulation. The pseudocode used for this algorithm can be read in the previous chapter in Algorithm 2. With this algorithm, two types of figures are generated with each simulated run: an occupancy grid and a convex hull of all the photoresistor's sensed points. The occupancy grid depicts free space, detected edges from the photoresistor, and detected gaps from the ultrasonic sensor. The convex hull is a boundary created by all the sensed photoresistor points, and it is created at the end of data collection – as such, it is not information used by the robots during the run. However, it depicts the information that can be given to future robots, allowing for future robots to have safer navigation through the previously completely unknown area.

Presented in Figures 4.2 and 4.3 are the occupancy grid and convex hull results of the algorithm on 4 different platforms: a circular platform with a radius of 15 centimeters (r=15 cm), a regular hexagon with a minor radius of 15 centimeters (r_{min}

= 15 cm), an ellipse with a minor radius of 15 centimeters ($r_{min} = 15$), and a circular platform with a radius of 25 centimeters (r = 15 cm) in order to show the scalability of the algorithm on a larger platform.

In the figures below, the sensing algorithms for exploration are run until the robot has closed a loop on its path or has fallen off the platform, at which point the algorithm is manually terminated and the final figures are generated from the mapped areas.

4.1.3 Linear Constraint Results

In this section, the maps from the Parking algorithm are presented based on their performance in simulation. The pseudocode for this algorithm can be seen in Algorithm 3. In addition to the same two types of figures shown in the Parking Results section (the occupancy grids as seen in Figure 4.4 and the convex hulls as seen in 4.5, in this section we also present a third type of figure as seen in Figure 4.6 in order to show the linear constraints that the robot created during traversal. The results of this algorithm are again presented on the same four simulation environment platforms: the r = 15 cm circle, the $r_{min} = 15$ cm regular hexagon, the $r_{min} = 15$ cm oval, and the r = 25 cm circle.

Ideally, the control point of the robot should be placed at the center of mass, as this location is the main determining factor in whether the robot will fall off a platform. With Puzzlebots, this center of mass point is at the center of the two wheels. However, a singularity is encountered with this center of mass as the control point when trying to constrain the robot's motion within the desired linear constraints. Because the angular velocity of the robot has no effect on the translational component on the center of the robot, the center of the robot is only able to move in the x-direction during one instantaneously small timestep according to the transition equation. Thus, if the center of mass is used as the control point, the minimization problem as defined in Equation 3.6 will only return an angular velocity, $\omega = u[1]$, value that minimizes the objective function with regards to $\omega^* = u^*[1]$, instead of returning an angular velocity that will help the robot avoid the unsafe zone.

In order to circumnavigate this singularity, the control point for this algorithm was moved to be a d distance in front of the center of mass, which is a location whose

y position can be altered in one timestep. This control point is the location of the robot used to detect if the robot is violating any constraints. The d value discussed here is shown in the Jacobian as seen in Equation 3.5.

In Subfigure 4.5c it is important to note that unlike the trajectory of the robot from Subfigure 4.3c from the Parking Algorithm, the robot does not fall off the oval with the Linear Constraints algorithm because of the added safety provided by the algorithm. However, it is important to note that the Linear Constraints Algorithm still does not offer any safety guarantees, as can be seen in Subfigure 4.5d where the robot still falls off the larger circle. On the other hand, the safety constraints are still helpful as the robot is still able to map a lot more of the large circle than when the Parking Algorithm traversed the same platform in Subfigure 4.3d. The cause of the robot falling off is due to the challenges as described in Subsection 3.3.4.1 of the previous chapter. Additionally, despite linear constraints offering safety, they also present new challenges as can be seen in Subfigure 4.6b, where one of the linear constraints cuts through a large area of the hexagon, and in an effort to abide by the constraint, the robot does not sense any points on the entire upper left portion of the platform.

4.1.4 Simulation Results Discussion

Example videos showing the demonstration of these algorithms can be seen at the following hyperlink: Simulation Videos.

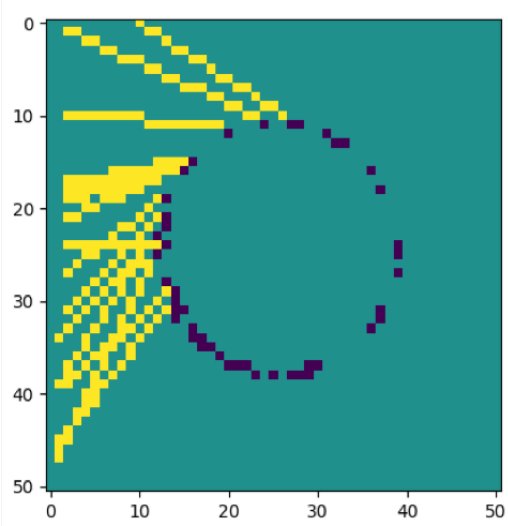
The basic algorithm provides the least amount of information about the environment, but is arguably the safest of the three presented algorithms, due to its full stop of motion when a gap is detected. If the robot stops moving whenever a gap is detected, the robot can be guaranteed to not fall off the platform, assuming the sensor is correctly identifying the difference between a gap and the platform properly.

The Parking Algorithm and the Linear Constraints Algorithm, each have their own advantages and disadvantages, depending on the ultimate goal of the robot. The Parking Algorithm is more likely to cause the robot to fall off the platform than the Linear Constraints Algorithm, as can be seen in the differences between Figures 4.3 and 4.5. However, the sensed points as shown on the occupancy grids are more dense for the Parking Algorithm and are better at representing the true shape of the edge.

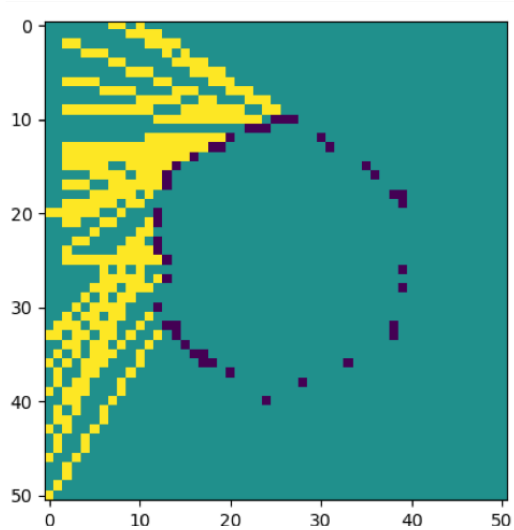
The sensed points are more sparse in the Linear Constraints Algorithm because in trying to abide by the linear constraints, the robot steers further away from the edges that it would need to ideally traverse in order to decrease the chances of falling off the platform.

The linear constraints can also make it more challenging for the robot to traverse all areas of the platform in scenarios where the linear constraints cut off large areas of the platform. However, in this scenario, the convex hulls and occupancy grids that would be given to the next robot for use would still be valid – they still have the ability to generate safe motion plans, though they would be more conservative in safety and require more robots to cross the gap than truly needed. Thus, this algorithm would be best used in scenarios where there is a prioritization of saving robots over mapping accuracy. On the other hand, the Parking Algorithm is better used in situations where the user would like to collect the most accurate edge data, with less regard for the safety of the robots.

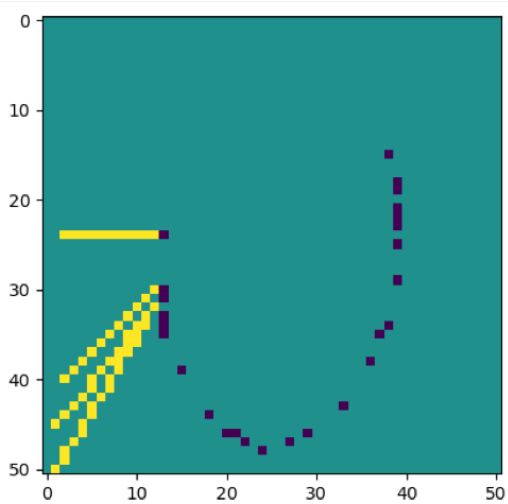
Ultimately, both the Parking Algorithm and the Linear Constraints algorithms are capable of mapping either a portion or the entirety of their provided environments. Despite both results showing the possibility of the robot falling off of the platform, and thus not guaranteeing safety, the goal of these low-cost robots is not to navigate their environment with safety guarantees. They are designed with low-cost sensors in order to facilitate the ability to map an environment without concern for high monetary loss, and with the algorithms presented here, the data collected from the robots provide information about the environment that will make future traversals safer.



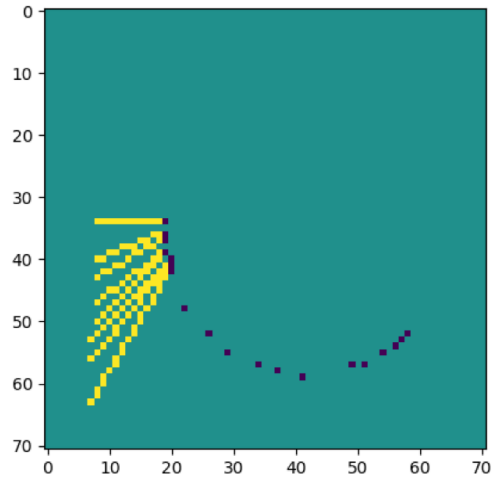
(a) Circle Platform



(b) Hexagon Platform

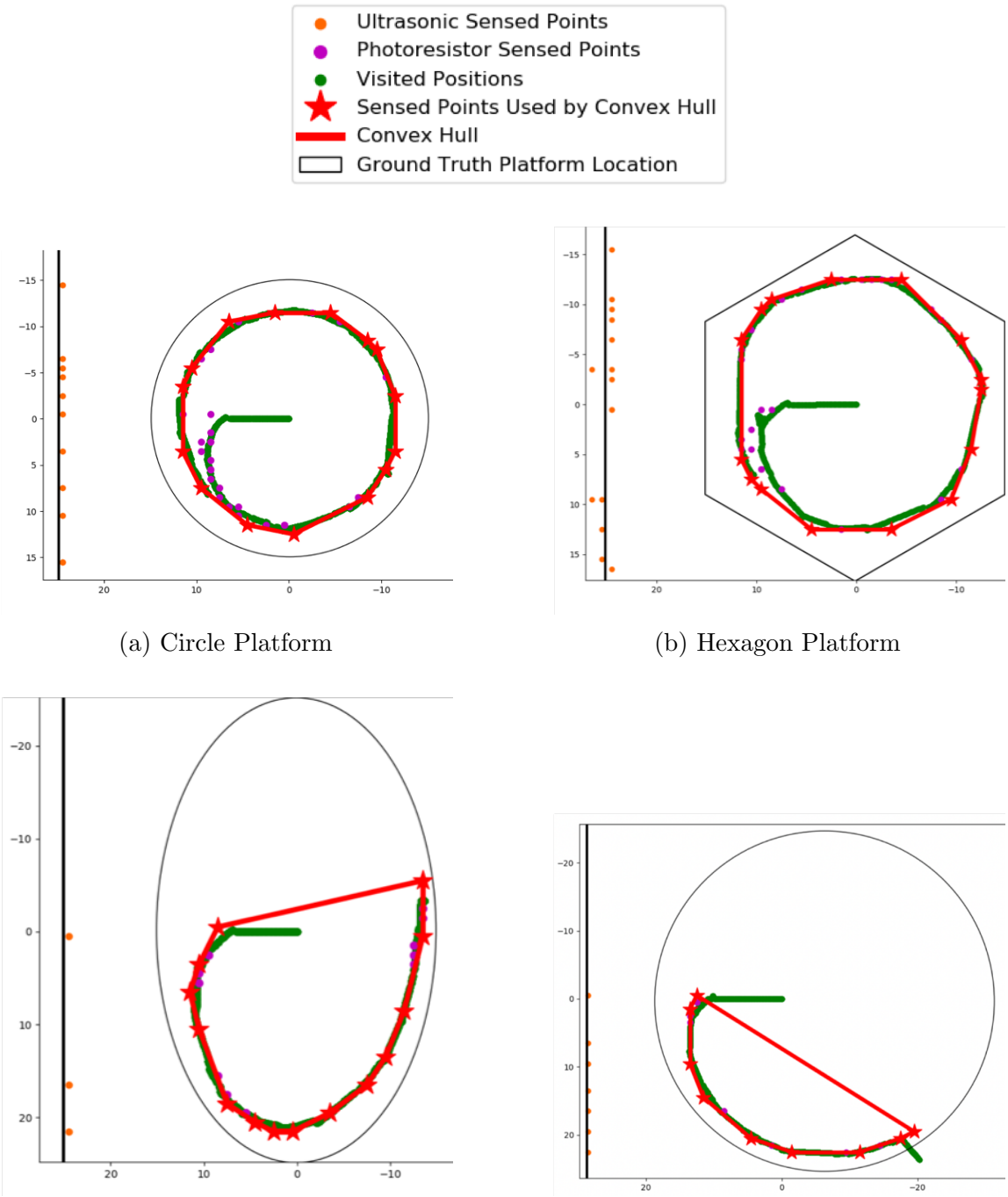


(c) Oval Platform



(d) Larger Circle Platform

Figure 4.2: Occupancy grid outputs for the parking algorithm in simulation after the robots have completed a closed loop path or have fallen off the platform. Each pixel represents 1 cm x 1 cm of environment area, where teal represents freespace, yellow represents gaps sensed via ultrasonic sensor, and purple represents gaps sensed via photoresistor. The shortest gap distance between the two platforms is 10 centimeters, and the true size of the platforms in Subfigures (a), (b), and (c) are all 30 centimeters in width, while the platform in (d) has a width of 50 centimeters.



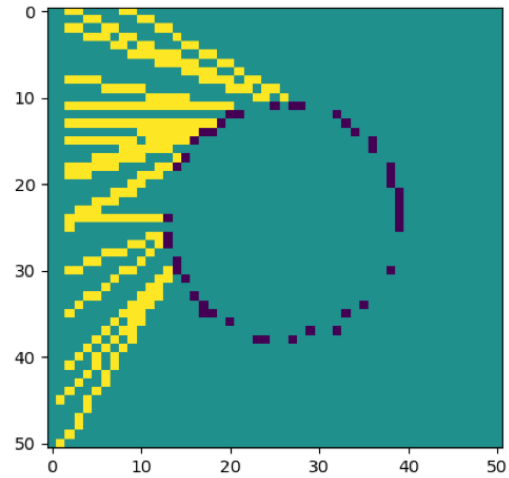
(a) Circle Platform

(b) Hexagon Platform

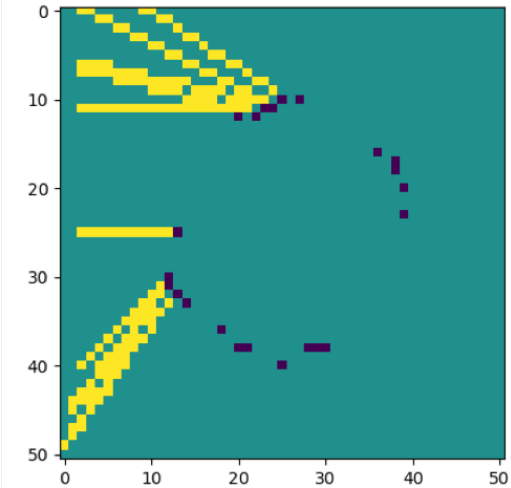
(c) Oval Platform

(d) Larger Circle Platform

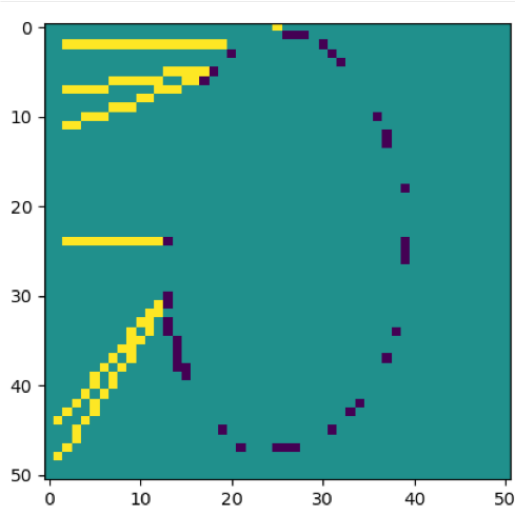
Figure 4.3: These are the convex hulls created by the robot with the Parking Algorithm after one closed loop path or after the robot has fallen off. The ultrasonic sensed points are the locations where the robot senses a second platform. The green plotted points are all the visited locations of the robot based on the center of the robot. In Subfigure (c) the robot gets stuck on the edge of the platform in a way where it is unable to traverse any further. In Subfigure (d) the robot fully falls off the platform. This is because since this algorithm has no safety benefits, it can sometimes enter unsafe states where it is unable to traverse further.



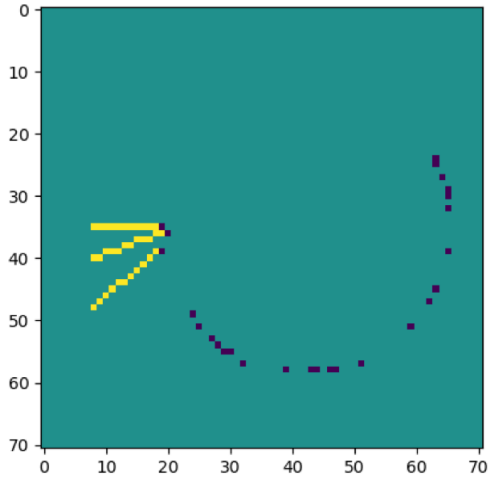
(a) Circle Platform



(b) Hexagon Platform

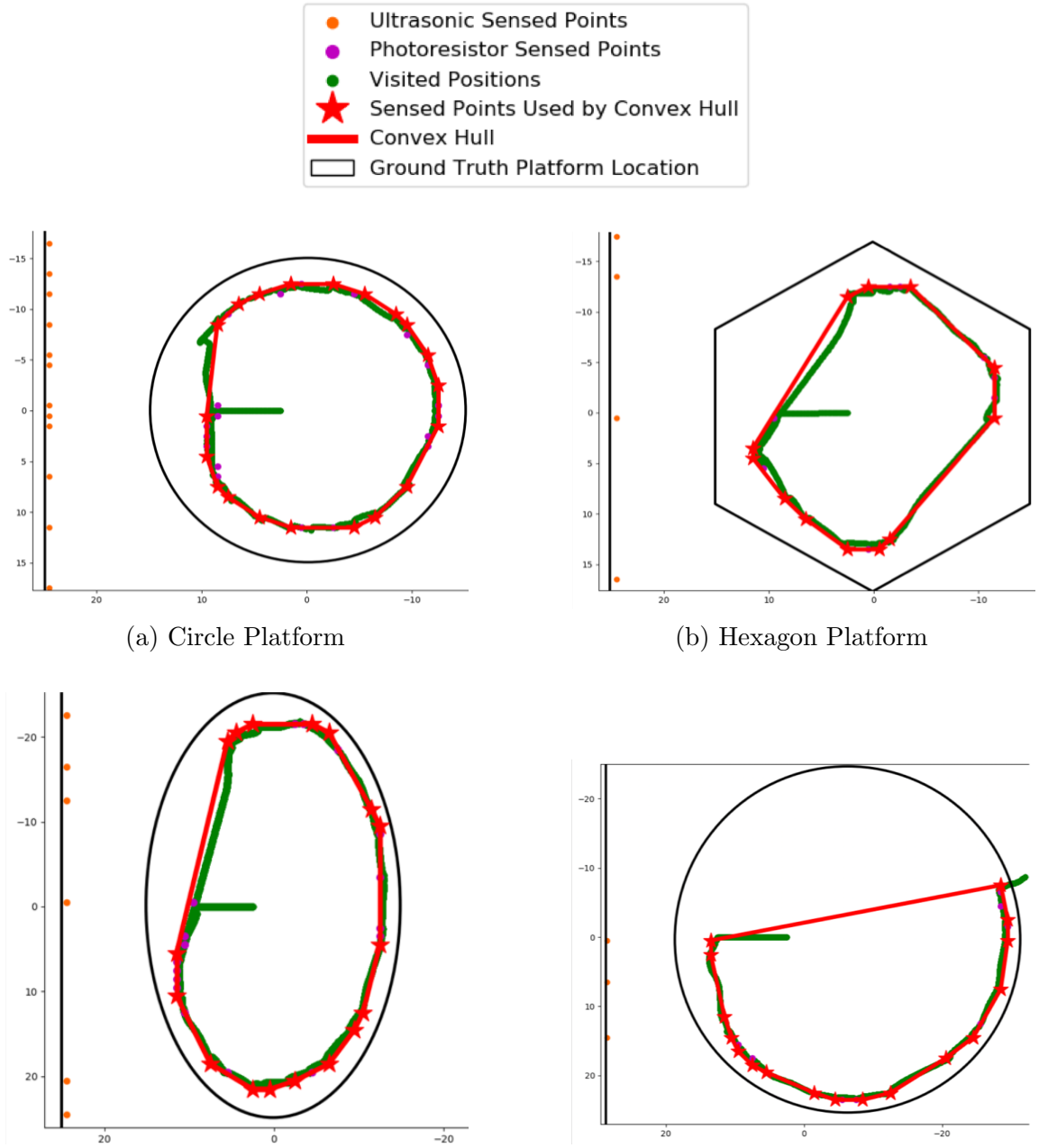


(c) Oval Platform



(d) Larger Circle Platform

Figure 4.4: Occupancy grid outputs for the Linear Constraints Algorithm in simulation after the robots have completed a closed loop path or have fallen off the platform. Each pixel represents 1 cm x 1 cm of environment area, where teal represents freespace, yellow represents gaps sensed via ultrasonic sensor, and purple represents gaps sensed via photoresistor. The shortest gap distance between the two platforms is 10 centimeters, and the true size of the platforms in subfigures (a), (b), and (c) are all 30 centimeters in width, while the platform in (d) has a width of 50 centimeters.



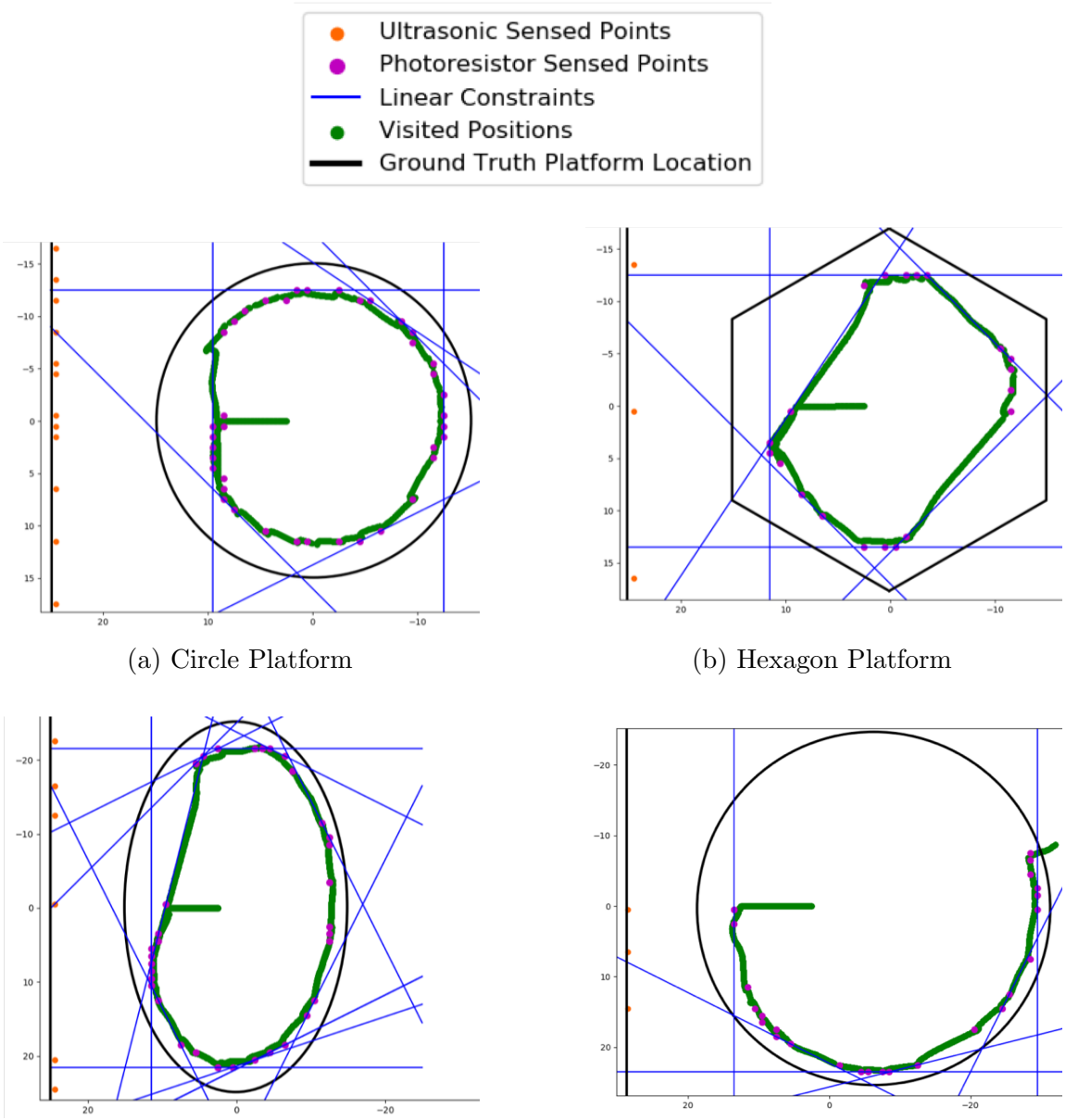
(a) Circle Platform

(b) Hexagon Platform

(c) Oval Platform

(d) Larger Circle Platform

Figure 4.5: These are the convex hulls created by the robot with the Linear Constraints Algorithm after one closed loop path or after the robot has fallen off. The ultrasonic sensed points are the locations where the robot senses a second platform. The green plotted points are all the visited locations of the robot based on the control point of the robot that is set to be in front of the center of mass.



(a) Circle Platform

(b) Hexagon Platform

(c) Oval Platform

(d) Big Circle Platform.

Figure 4.6: These are the linear constraints created by the robot with the Linear Constraints Algorithm. As the robot senses new edge points along the platform, it creates these linear constraints, and the robot solves the minimization in Equation (6) in an attempt to find a velocity that keeps the robot inside the constraints. The green plotted points are all the visited locations of the robot based on the control point of the robot that is set to be in front of the center of mass.

4.2 Hardware Demonstration

We demonstrate the basic stop on hardware to prove the working integration of the sensors onto the PuzzleBots system while maintaining other functionality. This can be seen in Figure 4.7. In this demonstration, the hardware stop is coded directly onboard the robot and is unable to do any mapping due to the current hardware limitations, as discussed in the next section. However, this hardware demonstration is still imperative to show that the robot is able to run with all sensors powered and to show the new safety capability that the photoresistor adds. A video of this demonstration can be seen by clicking the following hyperlink: Hardware Demonstration Video

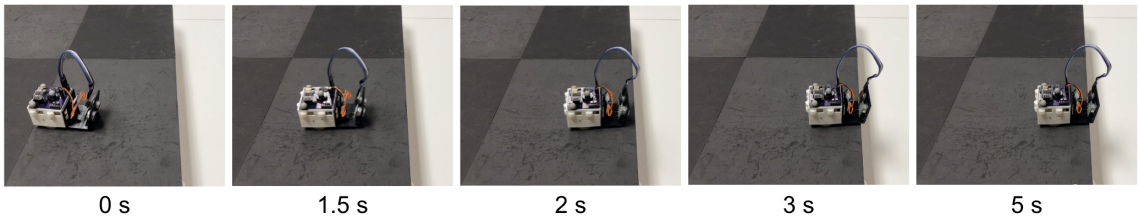


Figure 4.7: Screenshots of the robot's motion showing that the robot can use the photoresistor to stop at the edge in hardware.

4.2.1 Hardware Limitations

The intended design of the current hardware system is as follows. The planning implementations are to be done off-board so that all position data can be collected via an off-board motion capture system. This position data is necessary for mapping and planning. Therefore, in the hardware system, the robot only needs to handle the collection of sensor data so that it can send the data through a WiFi module to a nearby computer. Then, the sensor details are received by a Send/Receive script on the computer, which is connected to Robot Operating System (ROS). ROS sends the sensor data to a separate planning script that computes the desired velocity based on the received sensor data and position data collected from a motion capture system. The planner then sends the desired velocity back through ROS to the Send/Receive script, which then in turn sends the velocity signal back to the WiFi module onboard the robot. A system diagram detailing this interaction between hardware and software

can be seen in Figure 4.8.

During testing, it was found that the WiFi connection was too unstable for consistent communication of sensor data and velocity commands when the goal is to use the sensor data to prevent the robot from falling off the edge. If communication ever faltered, for example, then the velocity that had already been sent to the robot would persist, even in the case that sensor values indicating an edge are collected. Because of this, the hardware demonstration of the basic stop shown above was done by coding the stopping condition directly on the PCB (no motion capture data, WiFi, or ROS is used). Therefore, since the robot has no pose estimation capabilities, it is unable to create maps with the current hardware challenges. Future work should therefore prioritize adding pose estimation onboard the robot so that reaction to sensor data can be stabilized and allow for mapping with the more complex algorithms.

Furthermore, because the ultrasonic sensor is only used for mapping purposes, a demonstration of the ultrasonic's mapping ability on hardware cannot be seen in the above hardware demonstration. However, the ultrasonic sensor was still connected to the robot and powered during the demonstration, in order to show that the full system can function on 3V of battery.

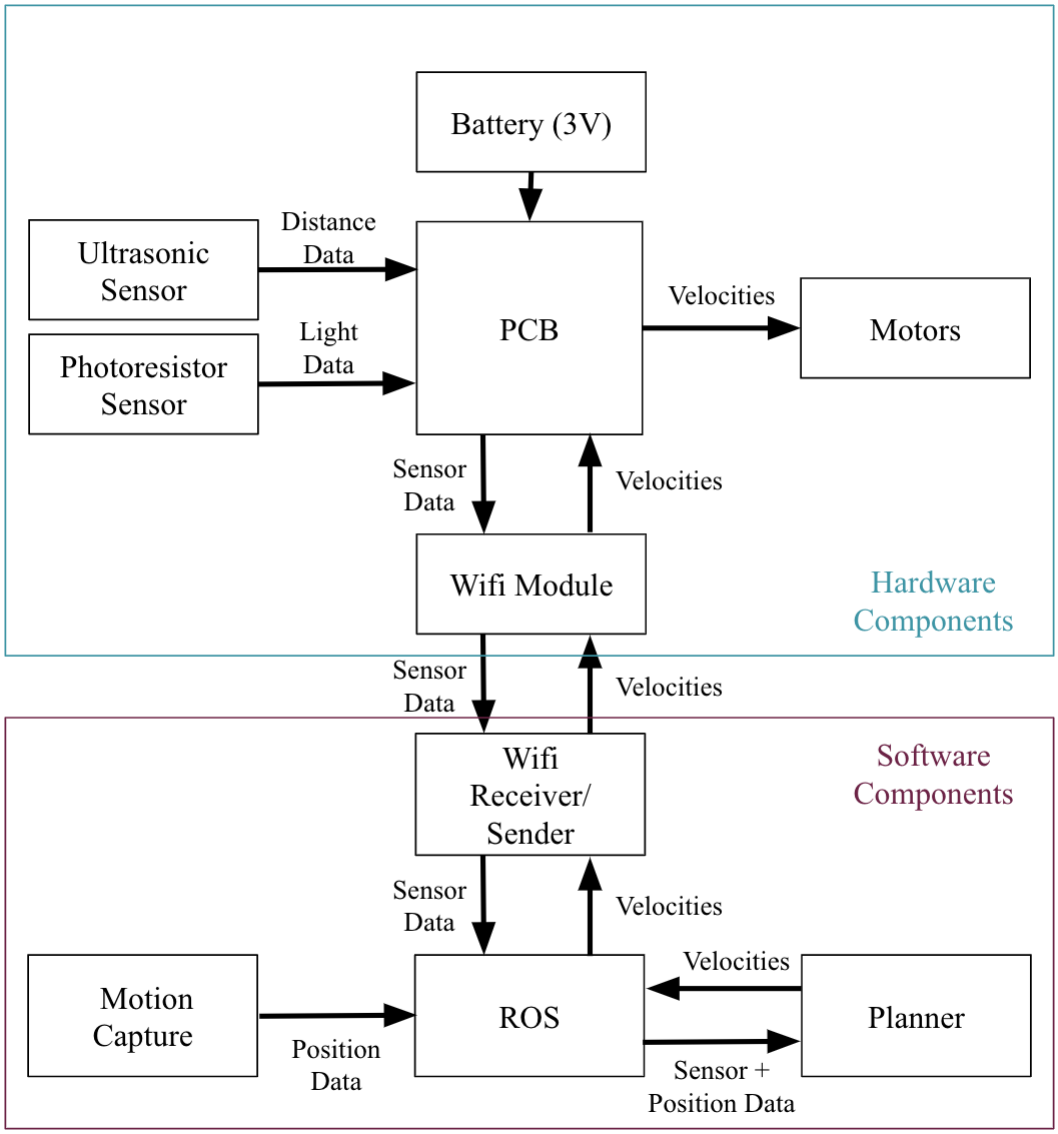


Figure 4.8: System Diagram showing how the hardware is designed to integrate with the software.

Chapter 5

Conclusions

In this thesis, we presented an approach to gap detection by implementing low-cost sensors onto the existing Puzzlebots chassis. By implementing a photoresistor, the robot is able to detect the amount of ambient light in the room – by angling this photoresistor toward the ground, the robot is able to detect the large changes in ambient light that occur when it approaches an edge due to the photoresistor receiving more exposure. With an ultrasonic sensor, the robot gains the ability to detect the length of the gap, giving future motion planning of the robots the ability to determine how many robots are needed to cross a gap. All the work presented in this thesis was implemented with regards to the Puzzlebots platform, but because both of these sensors are off-the-shelf components and were integrated onto the Puzzlebots robot for an additional cost of only $7.65, the work in this thesis can be easily extended to work with other small mobile robots.

After the implementation of both sensors onto the robots, we presented three algorithms of increasing complexity that allowed the robots to detect the gap and map full or partial convex platform edges. The first algorithm, the Basic Stop Algorithm, demonstrated the importance of having the sensors onboard the robot – with no sensors, the robot would fall off the edge in an unknown environment. The Parking and Linear Constraints Algorithms demonstrated a more advanced gap detection ability, giving the robot the ability to traverse the edge of convex platforms. While the Parking Algorithm doesn't have any built-in safety, the motion of the robot brings it closer to the edge, and the maps built with this algorithm are denser in

data collection and more true to the real platform shape. On the other hand, the Linear Constraint Algorithm provides an additional factor of safety, but sparser data collection points.

Both these second and third algorithms allow the robot to collect previously unknown information about the environment. Even in the event that the robot falls off the platform during mapping, the low-cost nature of the robot allows for the collection of the mapping information to be a worthwhile trade-off in SaR applications.

The prime limitation worth noting is the robot's inability to do pose estimation. Currently in the work presented, all pose is absolutely known in simulation. The most obvious first step is to implement sensors onboard the robot that can do basic pose estimation, so that pose can be known during field application use. Adding pose estimation to the robot will also allow for more complex hardware demonstrations of the algorithms presented in this thesis. The easiest way to do pose estimation would be through wheel encoders or using an Inertial Measurement Unit (IMU) to measure velocity. Both of these methods would introduce a large factor of error into the mapping problem, and more complex probabilistic models would have to be used for gap detection. Adding encoders or an IMU to the robot may require significant mechanical chassis redesign as the current robot only supplies 3 V and is just big enough to house all the electronics. Motors with encoders and IMUs will add significant expense and size requirements to the Puzzlebots Chassis. Furthermore, they both may also require higher voltage requirements to operate at an optimal level. In the future, this work could also be extended for use in multi-agent systems. By using multiple robots at a time to map the environment, data collection could provide higher precision maps at a faster rate.

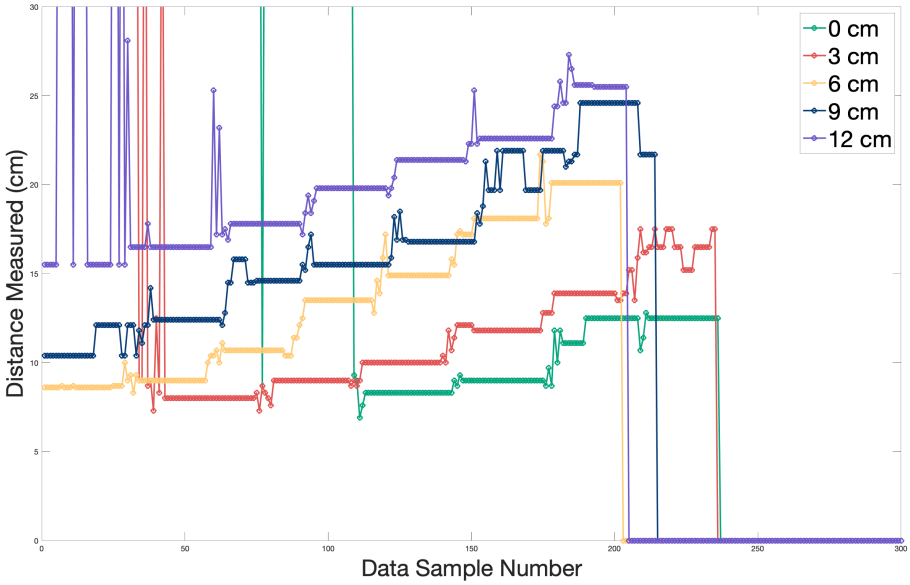
Appendix A

Appendix

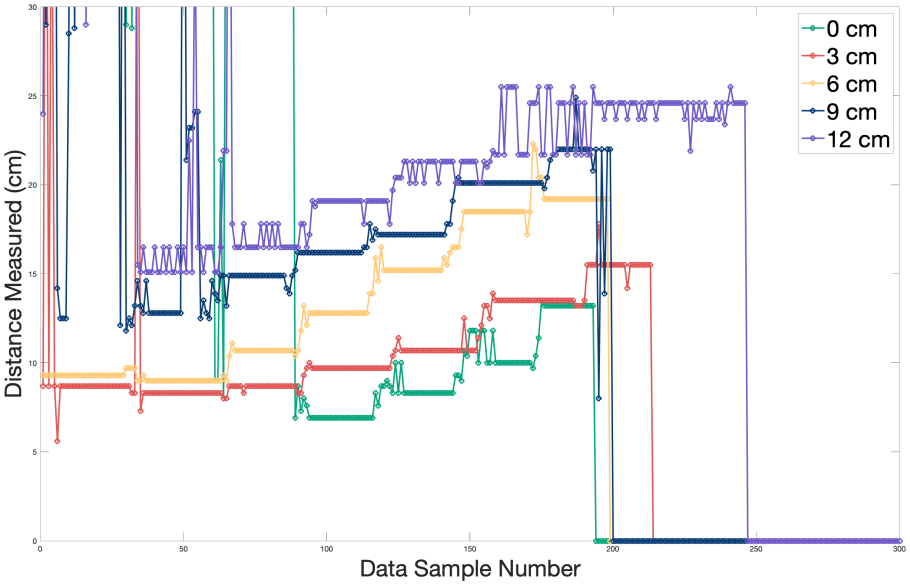
The graphs in Figures A.1 and A.2 show the results of the other ultrasonic sensor mounts that can be compared to Figure 2.7 – they show the raw data and the post processed data for the sensor mounts of angles 10°, 15°, 20° and 25°.

As seen in the raw data in Figure A.1, the plateau values are much less noisy and are much more distinguishable for the 10° mounting orientation compared to the plateaus of the other sensor mounting angles.

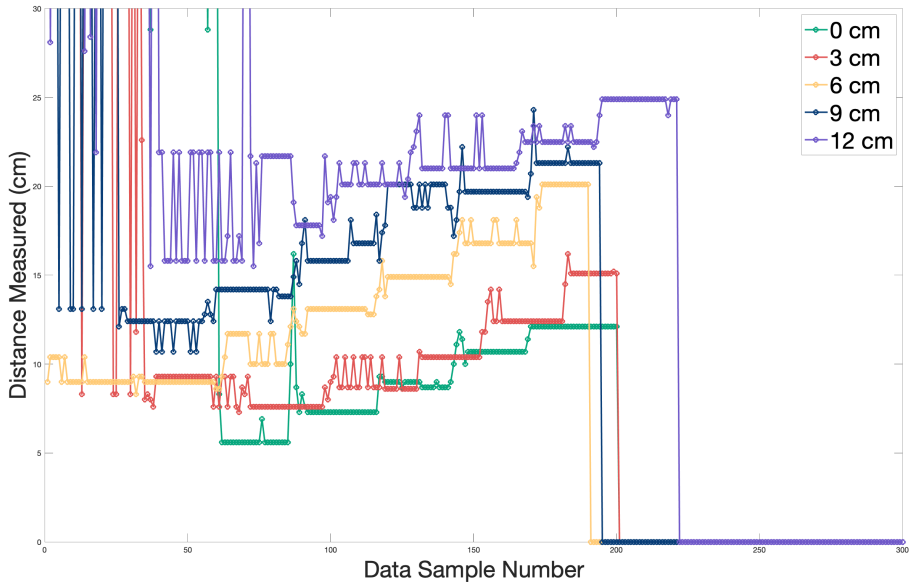
As can be seen in Figure A.2, the data points for the 10° mount are not the most accurate in their comparison between the ideal measurement and their actual measurement. However, because there is less variability between the data points and the ideal values over d and x parameters, we chose the 10° mount. The ideal variation would be for each x value data line to be fully linear – and by this standard, the 10° results are best.



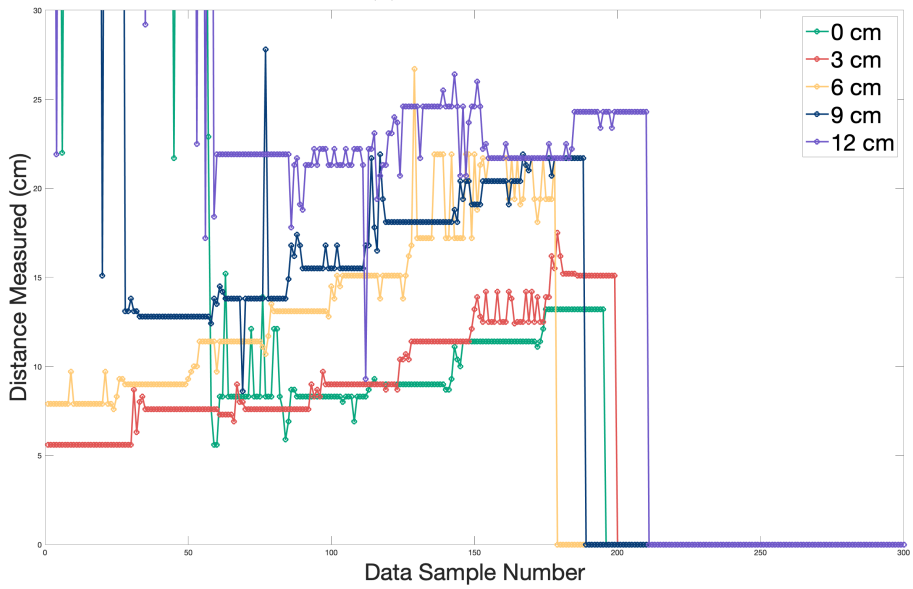
(a) 10 Degrees



(b) 15 Degrees

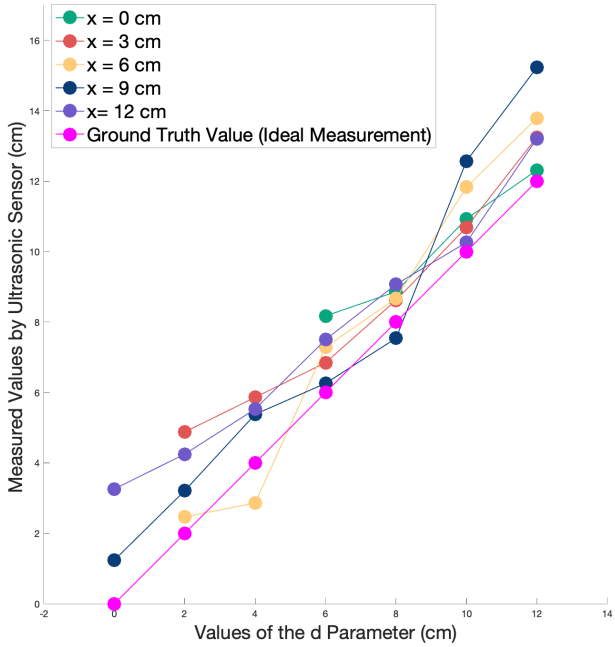


(c) 20 Degrees

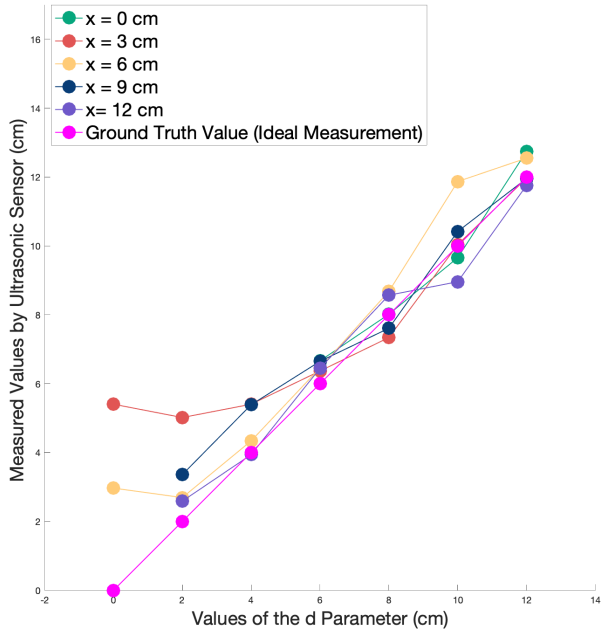


(d) 25 Degrees

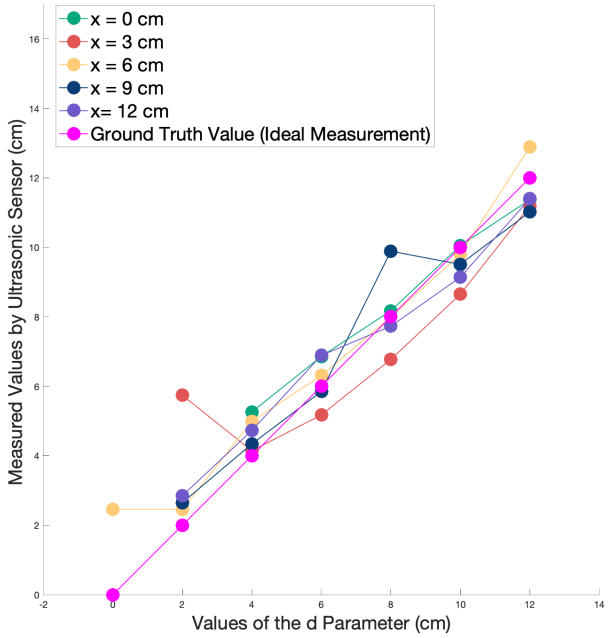
Figure A.1: The raw data for the second set of Ultrasonic Sensor experiments with different mounting orientations. The data points in the post-processed data seen in Figures 2.7 and A.2 are extracted from a single data point in the plateaus of the raw data. Each set of color lines represents the x parameter value as shown in Table 2.1 where each colored data line in each graph represents a collection of data from one experimental run. For example, each data point corresponding to sample number 200 is a single data point corresponding to the 200th collection sample in a single ultrasonic sensor data stream collected.


x = 0 cm
x = 3 cm
x = 6 cm
x = 9 cm
x= 12 cm
Ground Truth Value (Ideal Measurement)
Measured Values by Ultrasonic Sensor (cm)
Values of the d Parameter (cm)

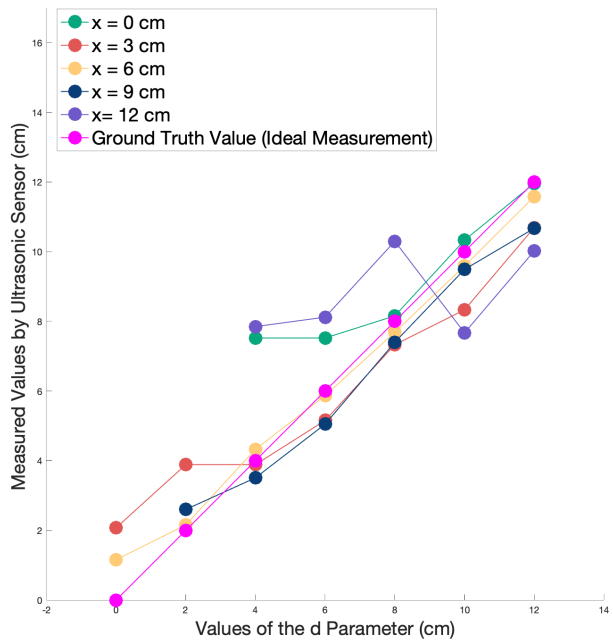
(a) 10°


x = 0 cm
x = 3 cm
x = 6 cm
x = 9 cm
x= 12 cm
Ground Truth Value (Ideal Measurement)
Measured Values by Ultrasonic Sensor (cm)
Values of the d Parameter (cm)

(b) 15°



(c) 20°



(d) 25°

Figure A.2: The post-processed graphs for the angle mounts for the second ultrasonic characterization experiments. These point values are taken from the plateaus of the raw data shown in Figure A.1 and are corrected to show only the horizontal distance of the gap after accounting for the sensor angle and the x parameter value. The lines between points are shown here to more clearly display which data points are from the same x parameter group. The parameters of this experiment are shown in Table 2.1.

Bibliography

[1] Farshad Arvin, Khairulmizam Samsudin, and Abdul Ramli. Development of a miniature robot for swarm robotic application. *International Journal of Computer and Electrical Engineering*, 1, 01 2009. doi: 10.7763/IJCEE.2009.V1.67.

[2] Farshad Arvin, John Murray, Chun Zhang, and Shigang Yue. Colias: An autonomous micro robot for swarm robotic applications. *International Journal of Advanced Robotic Systems*, 11(7):113, 2014. doi: 10.5772/58730. URL https://doi.org/10.5772/58730.

[3] M Zafri Baharuddin, Izham Z Abidin, S Sulaiman Kaja Mohideen, Yap Keem Siah, and Jeffrey Tan Too Chuan. Analysis of line sensor configuration for the advanced line follower robot. *University Tenaga Nasional*, 2005.

[4] Vicky Barua, Mohamad Arif, Shahid Uddin, Mithun Das, Mohammad Shafiul, Nazmun Nahar, and Abhijit Pathak. An ultrasonic line follower robot to detect obstacles and edges for industrial and rescue operations. *International Journal of Computer Applications*, 176:31–37, 2020.

[5] David Brandt, David Johan Christensen, and Henrik Hautop Lund. Atron robots: Versatility from self-reconfigurable modules. In *2007 International Conference on Mechatronics and Automation*, pages 26–32, 2007. doi: 10.1109/ICMA.2007.4303511.

[6] Daniel S. Drew. Multi-agent systems for search and rescue applications. *Defense, Military, and Surveillance Robotic*, 2021.

[7] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22:46 – 57, 07 1989. doi: 10.1109/2.30720.

[8] Giovanni C. Pettinaro Francesco Mondada and Andre Guignard et al. Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17:193–221, 2004. doi: https://doi.org/10.1023/B:AURO.0000033972.50769.1c.

[9] Simon Garnier, Faben Tache, Maud Combe, Anne Grimal, and Guy Theraulaz. Alice in pheromone land: An experimental setup for the study of ant-like robots. In *2007 IEEE Swarm Intelligence Symposium*, pages 37–44, 2007. doi: 10.1109/SIS.2007.368024.

[10] Paulo Gonçalves, Paulo Torres, Carlos Alves, Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1, 01 2009.

[11] Bahar Haghighat, Emmanuel Droz, and A. Martinoli. Lily: A miniature floating robotic platform for programmable stochastic self-assembly. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015:1941–1948, 06 2015. doi: 10.1109/ICRA.2015.7139452.

[12] Vivek Hanumante, Sahadev Roy, and Santanu Maity. Low cost obstacle avoidance robot. *International Journal of Soft Computing and Engineering (IJSCE)*, 3: 52–55, 01 2013. doi: 10.6084/M9.FIGSHARE.1327873.

[13] Hasith Karunasekera, Handuo Zhang, Tao Xi, and Han Wang. Stereo vision based negative obstacle detection. In *2017 13th IEEE International Conference on Control Automation (ICCA)*, pages 834–838, 2017. doi: 10.1109/ICCA.2017.8003168.

[14] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511546877.

[15] Guanqi Liang, Haobo Luo, Ming Li, Huihuan Qian, and Tin Lun Lam. Freebot: A freeform modular self-reconfigurable robot with arbitrary connection point - design and implementation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6506–6513, 2020. doi: 10.1109/IROS45743.2020.9341129.

[16] Chao Liu, Qian Lin, Hyun Kim, and Mark Yim. SMORES-EP, a modular robot with parallel self-assembly. *Autonomous Robots*, 47(2):211–228, dec 2022. doi: 10.1007/s10514-022-10078-1. URL https://doi.org/10.1007%2Fs10514-022-10078-1.

[17] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. In Alícia Casals, editor, *Sensor Devices and Systems for Robotics*, pages 253–276, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[18] Jerker Nordh and Karl Berntorp. Extending the occupancy grid concept for low-cost sensor-based slam. *IFAC Proceedings Volumes*, 45 (22):151–156, 2012. ISSN 1474-6670. doi: https://doi.org/10.3182/20120905-3-HR-2030.00079. URL https://www.sciencedirect.com/science/article/pii/S1474667016336035. 10th IFAC Symposium on Robot Control.

[19] Mehran Pakdaman and M. Mehdi Sanaatiyan. Design and implementation of line follower robot. In *2009 Second International Conference on Computer and Electrical Engineering*, volume 2, pages 585–590, 2009. doi: 10.1109/ICCEE.

2009.43.

[20] Sujni Paul and C. Beulah Christalin Latha. Shortest path traversal in a maze with wall following robot. *AIP Conference Proceedings*, 2670(1):030002, 2022. doi: 10.1063/5.0116117. URL https://aip.scitation.org/doi/abs/10.1063/5.0116117.

[21] Maria Isabel Ribeiro. Obstacle avoidance. *Instituto de Sistemas e Robótica, Instituto Superio Técnico*, 1, 2005.

[22] John W. Romanishin, Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295, 2013. doi: 10.1109/IROS.2013.6696971.

[23] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2013.08.006. URL https://www.sciencedirect.com/science/article/pii/S0921889013001474. Reconfigurable Modular Robotics.

[24] M. Shimizu, A. Ishiguro, and T. Kawakatsu. Slimebot: A modular robot that exploits emergent phenomena. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2982–2987, 2005. doi: 10.1109/ROBOT.2005.1570567.

[25] Arnab Sinha and Panagiotis Papadakis. Mind the gap: detection and traversability analysis of terrain gaps using lidar for safe robot navigation. *Robotica*, 31(7):1085–1101, 2013. doi: 10.1017/S0263574713000349.

[26] Marc Szymanski, Tobias Breitling, Jörg Seyfried, and Heinz Wörn. Distributed shortest-path finding by a micro-robot swarm. In *ANTS Workshop*, 2006.

[27] Sebastian Thrun. Robotic mapping: a survey. 2003.

[28] Vito Trianni, Stefano Nolfi, and Marco Dorigo. Cooperative hole avoidance in a swarm-bot. *Robotics and Autonomous Systems*, 54(2):97–103, 2006. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2005.09.018. URL https://www.sciencedirect.com/science/article/pii/S0921889005001478. Intelligent Autonomous Systems.

[29] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997. doi: 10.1109/CIRA.1997.613851.

[30] T. Yata, L. Kleeman, and S. Yuta. Wall following using angle information mea-

sured by a single ultrasonic transducer. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 2, pages 1590–1596 vol.2, 1998. doi: 10.1109/ROBOT.1998.677372.

[31] Sha Yi, Zeynep Temel, and Katia Sycara. Puzzlebots: Physical coupling of robot swarms. In *Proceedings of (ICRA) International Conference on Robotics and Automation*, May 2021.

[32] Sha Yi, Zeynep Temel, and Katia Sycara. Configuration control for physical coupling of heterogeneous robot. In *Proceedings of (ICRA) International Conference on Robotics and Automation*, May 2022.

[33] Jing Zhou, Jiacheng Wang, Jiazhong He, Jian Gao, Aixi Yang, Sideng Hu, and Qiang Li. Design, fabrication, and control algorithm of self-reconfigurable modular intelligent vehicles. *Applied Sciences*, 12:6886, 07 2022. doi: 10.3390/app12146886.

[34] Djemel Ziou, Salvatore Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998.