

Spring 4-25-2019

## Autonomous Object Collecting and Depositing Robot

Erik Anderson

Follow this and additional works at: <https://digitalcommons.murraystate.edu/honorsthesis>



Part of the [Electrical and Computer Engineering Commons](#)

---

### Recommended Citation

Anderson, Erik, "Autonomous Object Collecting and Depositing Robot" (2019). *Honors College Theses*. 205.

<https://digitalcommons.murraystate.edu/honorsthesis/205>

This Thesis is brought to you for free and open access by the Student Works at Murray State's Digital Commons. It has been accepted for inclusion in Honors College Theses by an authorized administrator of Murray State's Digital Commons. For more information, please contact [msu.digitalcommons@murraystate.edu](mailto:msu.digitalcommons@murraystate.edu).

Murray State University Honors College

HONORS THESIS

Certificate of Approval

\*\*\*

Erik Johan Anderson

May 2019

Approved to fulfill the  
Requirements of HON 437

---

Dr. Aleck W. Leedy, Associate Professor  
Institute of Engineering

Approved to fulfill the  
Honors Thesis requirement  
Of the Murray State Honors  
Diploma

---

Dr. Warren Edminster, Executive Director  
Honors College

Examination Approval Page

Author: Erik Johan Anderson

Project Title: Autonomous Object Collecting and Depositing Robot

Department: Institute of Engineering

Date of Defense: April 25, 2019

Approval by Examining Committee:

---

Dr. Aleck W. Leedy, Advisor

---

Dr. Joshua Ridley, Defense Committee Member

---

Dr. Kevin Miller, Defense Committee Member

Autonomous Object Collecting and Depositing Robot

Submitted in partial fulfilment

of the requirements

for the Murray State University Honors Diploma

Erik Johan Anderson

May 2019

## **Table of Contents**

<b>I. INTRODUCTION AND JUSTIFICATION .....</b>	<b>1</b>
<b>II. RULES, SPECIFICATIONS, AND SCORES .....</b>	<b>2</b>
<b>III. PRELIMINARY ANALYSIS .....</b>	<b>2</b>
<b>IV. INITIAL SUBSYSTEM AND FUNCTIONAL ANALYSIS .....</b>	<b>6</b>
<b>A) CONTROL .....</b>	<b>6</b>
<b>B) COURSE DETECTION AND POSITION TRACKING.....</b>	<b>8</b>
<b>C) OBJECT RECOGNITION .....</b>	<b>17</b>
<b>D) OBJECT MANIPULATION .....</b>	<b>18</b>
<b>E) MOVEMENT .....</b>	<b>23</b>
<b>F) POWER .....</b>	<b>27</b>
<b>G) PHYSICAL STRUCTURE .....</b>	<b>28</b>
<b>V. CHANGES TO SUBSYSTEM AND FUNCTIONAL ANALYSIS.....</b>	<b>32</b>
<b>A) CONTROL .....</b>	<b>32</b>
<b>B) COURSE DETECTION AND POSITION TRACKING.....</b>	<b>34</b>
<b>C) OBJECT RECOGNITION .....</b>	<b>36</b>
<b>D) OBJECT MANIPULATION .....</b>	<b>41</b>
<b>E) MOVEMENT .....</b>	<b>43</b>
<b>F) POWER .....</b>	<b>45</b>
<b>G) PHYSICAL STRUCTURE .....</b>	<b>47</b>
<b>VI. CONTROL LOGIC .....</b>	<b>49</b>
<b>VII. CONCLUDING ANALYSIS .....</b>	<b>55</b>
<b>VIII. REFERENCES .....</b>	<b>60</b>

## **APPENDICES**

<b>A. RULES, SPECIFICATIONS, AND POINTS .....</b>	<b>A-1</b>
<b>B. ADDITIONAL FIGURES AND TABLES .....</b>	<b>A-5</b>
<b>C. FINAL COMPETITION SCORES.....</b>	<b>A-8</b>
<b>D. CALCULATIONS .....</b>	<b>A-12</b>
<b>E. CODE.....</b>	<b>A-13</b>

## **List of Tables and Figures**

<b>Tables</b>	<b>Page</b>
1 Comparison of Microcontrollers.....	7
2 Comparison of Motors .....	27

<b>Figures</b>	<b>Page</b>
1 Side view of bar that prevents spheres from rolling forward.....	23
2 Front view of robot. ....	30
3 Angled view of attempted spiked wheel. Hot glue was placed to try to increase traction, but ultimately decreased it. ....	45
4 Top view of final robot. ....	50
5 Block diagram of robot. ....	51
6 Rankings of 12 best-performing robots at the robotics competition. ....	57
7 Attachment A, the course setup diagram. ....	A-5
8 Attachment B, the competition ladder. ....	A-5
9 Blueprint of the course.....	A-7
10 Additional front view of robot. ....	A-8

11	Front view of final robot. ....	A-8
12	Scores of 1 <sup>st</sup> to 13 <sup>th</sup> highest scoring teams. ....	A-9
13	Scores of 14 <sup>th</sup> to 27 <sup>th</sup> highest scoring teams. ....	A-10
14	Scores of 28 <sup>th</sup> to 40 <sup>th</sup> highest scoring teams. ....	A-11
15	Scores of 42 <sup>nd</sup> highest scoring teams. ....	A-11



## Abstract

The Institute of Electrical and Electronics Engineers (IEEE) hosted SoutheastCon 2019, with one of the conference's focuses being a robotics competition. The robotics competition simulates the task of removing space debris near a space station and depositing it in predetermined areas. This task must be completed by an autonomous robot that meets a variety of physical design criteria. The performance of robots is measured by means of points, wherein accomplishing positive tasks earns points and vice versa, and the robot that most consistently scores points wins.

In response to the competition, the four engineering physics majors created a robot that uses LiDAR and cameras to accomplish the goals put forth by IEEE. The LiDAR serves to detect the geometry of the course and allows the robot to maneuver without hitting course features, given known information concerning course geometry. At the same time, a camera enables the robot to detect where it is in the course by recording the tape colors found in the course, as these are unique to specific areas of the course. Another camera observes the colors of objects collected by the robot. Whenever the colors detected by the two cameras match sufficiently, the robot knows that it can deposit its collected objects and maximize its scored points. The robot runs on two Raspberry Pi computers and is coded in Python. It makes use of the open-source OpenCV library for auxiliary camera-based functions. The robot produced promising results, and a multitude of lessons were learned from its creation.

## **I. INTRODUCTION AND JUSTIFICATION**

The Institute of Electrical and Electronics Engineers (IEEE) hosts a variety of conferences throughout the world. Its third region, containing the southeastern United States, hosts an annual conference referred to as SoutheastCon. In this conference, a multitude of presentations, workshops, and competitions are hosted. Specifically, one of these competitions is a student hardware competition, which typically takes the form of a robotics competition. The premise and objectives of this competition varies year to year. For SoutheastCon 2019, the robotics competition premise as described by IEEE was as follows:

“The repercussions of a collision with even a tiny piece [of] debris traveling at orbital or interplanetary velocities could be devastating; such a collision could destroy a spacecraft. The cleanup process is arduous and expensive, and international regulations only permit unmanned, autonomous bots to assist with the cleanup efforts. To reduce the risk of disaster, debris removal bots have been commissioned by the international community. These autonomous debris removal bots will assist with cleanup of the debris in Earth’s orbit. Your mission is to design such an autonomous debris removal bot. In the spirit of cooperation that led to our greatest innovations in space your bot must cooperate with other debris removal bots in orbit and thus making space travel safer for everyone [1, p. 3].”

This premise translates directly into the technical objective of the competition:

“To clear orbital space debris while avoiding Spacetels and return to home base within the time limit of three minutes. The score is determined by a number of

factors including: 1) the number of complete playing field orbits, 2) the number and type of space debris cleared, 3) sorting cleared debris, 4) returning to assigned corner square (home base), and 5) avoiding collision with Spacetel [1, p. 3].”

With both the premise and objective in mind, it ought to be noted that Murray State University hosts an IEEE student group that has consistently partaken in the robotics competition. The robot’s creators are members of this group. These individuals shared an interest in applying their skills to the robotics competition, and thus chose to create a submission to represent Murray State University at SoutheastCon 2019. Additionally, these individuals learned that the creation of this submission constituted a large enough endeavor that it could be represented as a senior design project. As such, this course of action was chosen to be more efficient, despite associated additional tasks and the need to minimize costs more aggressively than would otherwise be required. Finally, it was learned that the development of a submission to the robotics competition could be written about for this thesis, and thus this was done for the sake of efficiency.

## **II. RULES, SPECIFICATIONS, AND SCORING**

The SoutheastCon 2019 robotics competition was given an extensive set of rules and specifications that would serve as constraints and points of consideration with regards to the creation of a submission. Additionally, a scoring system was provided for determining which robots performed best of all robots submitted. This system was able to be used as a simple and germane system by which to gauge the performance of the robot described within this document. The entirety of these rules, specifications, and scoring metrics may be found in Appendix A.

### **III. PRELIMINARY ANALYSIS**

As previously stated, the entirety of the rules, specifications, and scoring systems are described within Appendix A. However, several rules and specifications had a guiding influence on the robot's function and design. With regards to the course's specifications, it was to consist of a square of walls measuring 9 feet to a side and 11.25 inches high, with a 2-meter-wide circle of tape centered inside. Centered inside that circle was a central structure measuring 9 inches to a side and 11.25 inches high, rotated 45 degrees relative to the bounding walls. The sides of this structure were to be painted with a specific order of colors – red, green, blue, and yellow. Positioned at the cardinal points of the bounding circle's circumference were 4 blinking lights, measuring 3 inches to a side and 3.8 inches high. A blueprint showcasing these physical dimensions may be seen in Figure 9 in Appendix B, with a more general image being found in Appendix A as figure 7. With regards to the earning of points, 8 cubes and 4 spheres, respectively measuring 2 inches and 2.5 inches to a side, were to be placed within the 2-meter-wide circle. These cubes and spheres were to be colored red, green, blue, and yellow, with there being 2 cubes and 1 sphere of each color. Ten points were to be awarded for each cube or sphere moved out of the 2-meter-wide circle. Another 10 points were to be awarded for each cube or sphere that was moved to any given corner of the course's bounding walls, and a final 10 points were to be awarded for each cube or sphere placed in a corner of its color. A corner's color was given by the color of the central structure's wall facing it. Five points were earned for each counter-clockwise orbit of the robot about the central structure, and 10 points were deducted for each time the robot collided with one of the four lights. Finally, points could be earned over the course of 3 minutes, with the time starting when the robot was activated by means of either button pressing or switch flipping.

With these rules and specifications in mind, three distinct approaches were considered to attempt to maximize the points that the robot could earn. The first approach considered was to create a robot that would maneuver the playing area by some method and move cubes and spheres to their color-appropriate corners, thereby maximizing the points earned through the manipulation of blocks and cubes. Specifically, each cube or sphere placed in its appropriate corner would earn 30 points – 10 from being removed from the circle, 10 from being placed in a corner, and 10 from being placed in a corner of its color. Given the presence of 12 cubes and spheres, this meant that 360 points could be earned through spheres and cubes, in addition to any points awarded for orbiting the central structure and the completion of additional tasks described in Appendix A. The second approach considered was to create a robot that would orbit the central structure as many times as possible within the previously described 3 minutes available to earn points, with no regard given to the collection of objects. Given that only 5 points would be awarded for each orbit of the central structure, this approach would require the robot to orbit the central structure 72 times to earn the same 360 points that the previous approach could earn. Further orbits would be required to match the points earned by the first approach whenever it happened to orbit the central structure. The third approach considered was a combination of the previous two approaches, wherein a robot would attempt to score points by means of removing objects from the circle and orbiting the central structure. Less importance would be placed on placing every object in its respectively colored corner or orbiting as much as possible within the given 3 minutes. Rather, objects would be removed from the course as opportunities to do so arose, and color matching would be practiced as opportunities permitted doing so easily.

Of the approaches considered, the implementation of the first approach was initially worked towards. The reasons for this were varied, although the two most notable ones were that

such an approach simultaneously permitted the greatest exploration of everyone's abilities and seemed to be truest to the spirit of the competition's premise. The first reason was the fact that the robot's developers believed that while they may not immediately possess the ability to create the robot described by the first approach, they would be able to gain the skills necessary to do so over the course of the fall semester of 2018 and the spring semester of 2019. Meanwhile, the second justification was founded on the idea that if the competition's scenario is based around the idea of removing debris from space, then it was most appropriate to focus the robot's functionality on removing debris from the playing field. This stood in opposition to the second approach, which could be considered in such light to be taking undue advantage of the scoring metrics used for the competition. However, it should be noted that over the course of the robot's creation, the third approach was ultimately pursued as a result of learning more about the limitations of the systems that were used. How this occurred will be discussed in later sections.

Once the course of action for designing the robot was known, analysis of the options available for implementing the robot had to occur. The analysis conducted was divided by the various subsystems that would eventually be found in the robot. These subsystems consisted of course and position tracking, object recognition, object manipulation, movement, control, power, and physical structure. The author was responsible for analysis of the course and position tracking, object manipulation, power, and physical structure subsystems, and thus can write at length about them. The author's associates were responsible for analysis of the other subsystems, and such analysis shall be summarized. It should be noted that the mentioned analysis occurred prior to any actual construction of the robot and implementation of the proposed systems. Once said activities began, the realities imposed by the products used and the skills of the robot's developers prompted additional analysis and changing of a variety of subsystems. These two sets

of analysis will be recognized in two distinct sections of this document, with initial analysis of subsystems being documented in section 4. Analysis of subsystem implementation in the robot, as well as any changes and analysis that occurred during such implementation, will be documented in section 5.

#### **IV. INITIAL SUBSYSTEM AND FUNCTIONAL ANALYSIS**

##### **a. Control**

To enable the robot to accomplish any tasks given to it, some form of control infrastructure had to be in place. This infrastructure would sensor data, analyze it, and produce outputs in response to the data that direct the action of motors and other devices. The composition of control infrastructure may vary widely based on the needs and purpose of a robot, and within a certain set of needs and purposes multiple options for the infrastructure may exist. With regards to the robot that was developed, it was determined that some form of microcontroller would enable the robot to fulfill its goals. Three microcontrollers were considered, with those being the Arduino Uno Rev 3, the Raspberry Pi 3 Model B, and the Sparkfun DEV-13610. The characteristics of the microcontrollers may be seen in Table 1.

*Table 1. Comparison of Microcontrollers*

Name	Processor Frequency	RAM	Number of I/O Pins	Number of USB Ports	Price
Arduino Uno	16 MHz	32 KB	14	1	\$14.95
Raspberry Pi 3 Model B	1.2 GHz	1 GB	40	4	\$35.00
Sparkfun DEV-13610	1.2 GHz	1 GB	0 (requires shields)	1	\$119.95

From the table, one may observe that the Raspberry Pi 3 Model B provides the same processor frequency and quantity of RAM as the Sparkfun DEV-13610, while costing a fraction of the price. These two characteristics are important because of their influence on the performance of the robot. Should the processor speed or amount of RAM be too low, some programs would slow down, behave erratically, or fail to function. Additionally, a higher processor frequency allows a greater quantity of incoming data to be processed. The Raspberry Pi 3 also provides the greatest number of I/O pins of the three microcontrollers, as well as the most USB ports. Because of this, multiple input devices or sensors could be connected to the microcontroller without needing to use supplementary circuit boards with the I/O pins or USB ports needed. With this in mind, a Raspberry Pi 3 Model B was selected as the primary microcontroller of the robot. Additionally, an Arduino Uno Rev 3 was selected as a complementary microcontroller, as previous years' experiences indicated that motor control ought to be handled through an Arduino Uno or similar microcontroller, rather than a Raspberry



Pi 3, to ensure that any erratic behavior on the part of the motors would not directly damage the main microcontroller.

For the robot to have been created and developed, a development environment had to have been selected. Given that a Raspberry Pi 3 Model B had been selected as the robot's primary microcontroller, it was possible for a full-fledged computer operating system to be loaded onto the microcontroller. As such, Linux was selected as the robot's operating system. This decision was made because of the fact that Linux is an open-source operating system with extensive documentation and a wide variety of implementations. Several of these implementations were made with efficiency in mind, meaning that more of the Raspberry Pi's resources would be available for processing data and controlling the robot if they were used. Thus, Raspbian – a version of Linux optimized for being run on Raspberry Pi microcontrollers – was chosen as the Raspberry Pi's operating system. With Raspbian permitting the ability to easily interface with the Raspberry Pi, the Robot Operating System was selected as the framework to develop the robot in. ROS was chosen because of its open-source nature and the fact that it was created for the express purpose of providing a framework of control for robots. In addition, the open-source nature of ROS meant that there were numerous packages available that could shorten development time if used, as they would ideally have allowed for advanced operations to be implemented relatively simply within the robot.

#### **b. Course Detection and Position Tracking**

To enable the robot to maneuver around the course without incident, some form of course detection and position tracking was required. Course detection was necessary to allow the robot to see the course's shape. In knowing the course's shape, as well as additional data collected by

the robot's sensors, the robot could plan or follow routes of travel through the course to accomplish its objectives. In addition to that, if the robot could detect the course relative to itself, then it would also be able to perform collision avoidance with regards to the course. It would thus avoid unnecessary damage to either the robot or its surroundings. Position tracking, as was to be implemented in the robot, was closely tied to course detection. If the robot could determine where the boundaries of the course were relative to itself, then it could know its position relative to the course. This, when combined with data regarding the positioning of objectives relative to the robot, would allow the robot to determine how it needed to travel to reach its objectives. There were many methods considered for how course detection and position tracking could be implemented in the robot, including ultrasonic sensing, odometry, camera-based feature recognition, and LiDAR.

Ultrasonic sensing possesses several attractive features with regards to implementation in the robot. Firstly, the ultrasonic sensors present on the consumer market are relatively cheap, typically costing between 10 and 50 dollars, although there are several products costing even less. While it is true that many ultrasonic sensors cost in excess of several hundred dollars, these are intended for precise measurement. Such precision would not be harmful to the robot's operation but would be unnecessary given that the robot only needs to detect when it starts approaching obstacles. As such, the use of ultrasonic sensors would enable a notable degree of cost saving in the robot's construction if large amounts of them are not needed. A second benefit of ultrasonic sensors is their low current draw. Among the previously referenced consumer ultrasonic sensors, current draw was typically found to be less than 5 mA at 5 V. This then means that if implemented, the amount of power consumed by the ultrasonic sensors would be very low compared to the power draw of all other included electronic devices. Finally, the data

output volume of the previously mentioned ultrasonic sensors may be considered non-existent compared to the other methods of course detection. This is said in confidence because none of the datasheets studied made note of a volume of data output, leading one to assume that any modern controlling system would be capable of processing ultrasonic sensor output without issue.

However, there are some detracting features of ultrasonic sensors. As noted by Sachin [2], ultrasonic sensors can have difficulty detecting surfaces that sit at an angle relative to an ultrasonic sensor. The result of this is that a robot using an ultrasonic sensor for course detection would be unable to detect walls located parallel to its sensor, thus preventing it from knowing whether it may collide with a wall to its side. This may be mitigated by using multiple differently orientated ultrasonic sensors, although this mitigation technique partially negates the benefit of ultrasonic sensors being low cost. In addition to that, because ultrasonic sensors calculate distance by means of Time of Flight calculations, they will only return the distance of the object that sits closest to them in the region that they detect. The consequence of this is that ultrasonic sensors cannot be used to obtain a detailed map of the robot's surroundings, which prevents in-depth consideration of the course. Finally, when multiple ultrasonic sensors are used simultaneously, their ultrasonic waves will interfere with one another. This causes inaccurate measurements of the robot's surroundings. Some ultrasonic sensors are designed with this possibility in mind and have methods of preventing this from occurring or mitigating its effect. However, these sensors are more expensive than their counterparts, which further negates the monetary advantage of using ultrasonic sensors.

Odometry is a method of tracking a robot's position by taking in data produced by movement features, such as motors and wheels, and calculating how far and in what direction it

has moved. More specifically, known data such as wheel travel distance and wheel separation is combined with measured data such as wheel rotation and robot heading. The formulae used to calculate a robot's position by odometry, as defined by Edwin Olson, can be seen in equations 1 to 5 [3]:

$$d_{center} = \frac{d_{left} + d_{right}}{2}, \quad (1)$$

$$\phi = \frac{d_{right} - d_{left}}{d_{baseline}}, \quad (2)$$

$$\theta' = \theta + \phi, \quad (3)$$

$$x' = x + d_{center} \cos(\theta), \text{ and} \quad (4)$$

$$y' = y + d_{center} \sin(\theta). \quad (5)$$

Odometry's main benefits would be the that its calculations are simple, the methods of taking its required measurements can be built into motors by way of encoders, and that it can easily complement any other method of course and position tracking because of its use of encoders. Encoders are devices that track the rotation of a motor shaft or attached wheel by using a disc with holes punched around its perimeter and a sensor that detects the passage of a hole. The sensor outputs a series of ones or zeroes whenever holes pass it, with the output quantity determining degree of rotation. There are multiple implementations of encoders available, although as noted by Olsen [3], quadrature phase encoders are recommended for odometry over mono phase encoders. This is because the former can distinguish between forward and backward motion and avoids the latter's inability to handle instances of rotation ending with a hole partially blocking the sensor. The problem with relying solely upon odometry for position tracking lies in the fact that any errors that enter its calculations promote the multiplication and

further accumulation of error. As such, while odometry may initially be accurate, it will quickly become inaccurate if its errors are not corrected through other forms of position tracking.

Another issue with using odometry for position tracking would be the fact that motors incorporating encoders are more expensive than similar motors that do not, which ultimately makes the robot more expensive than it might otherwise be. Of further note is the fact that odometry both incurs its own cost in motors and requires correction by other position tracking methods, which have their own cost.

Camera-based feature recognition refers to the idea of using one or more cameras to continuously take pictures of the course with the intent of applying algorithms that can identify visual features of the course and provide data on the course's shape that may be used for navigation. There are many varieties of such an approach to course detection, including calculating depth through binocular cameras, utilizing convolutional neural networks to classify course features, applying edge detection to identify notable course geometry, and more. The benefits of camera-based feature recognition include being versatile in their deployment when properly set up and being able to perform visual analysis of the course in a manner similar to how a human might.

However, the downsides of such course detection are also numerous. Firstly, any implementation utilizing neural networks will be hampered by the issue of neural networks requiring extensive training before they perform adequately. This training requires that the trainer have access to extensive datasets of images depicting course features that need to be classified, and the training itself can take upwards of several days to complete even when extensively parallelized. Such a case is discussed by Krizhevsky, Sutskever, and Hinton in their work on convolutional neural networks, wherein training of the network took five to six days [4].

Secondarily, while neural networks prove adept at classifying distinct features present in pictures and videos, for the purpose of course recognition the robot needs to be able to identify and differentiate between floors, different walls, course obstacles, and course objectives. While the latter two features are likely distinct enough that neural networks could be trained to recognize them, the former two are both large and uniform in composition, making it difficult to properly differentiate between them. Thirdly, the robot must be able to not only detect that a wall is a wall, but also how the wall is positioned relative to the robot. While depth detection with binocular cameras allows this, neural networks are not well suited to this task. Another downside would be that camera-based feature recognition in the forms considered is relatively computationally expensive. While this may not be an issue when running these processes on computers build with computational power in mind, the Raspberry Pi used to control the robot is not suited for computationally heavy tasks. As such, attempting to use camera-based feature recognition would likely lead to either excessive system resource use or reduced functionality due to a lack of available system resources. Finally, the cost of cameras compatible with the robot must be considered. In this instance, such cameras were found to typically range in price from 15 dollars to 70 dollars, depending on their resolution and whether they could record video. While this is not exorbitant, it should also be considered with respect to the computational intensity of camera-based feature recognition, in that such a method of course detection and position tracking would incentivize the use of more powerful and expensive computational hardware.

LiDAR is a method of surveying a target by means of shooting pulses of laser light at it, after which the time taken for the laser to return to its source and the wavelength of the returning laser may be used to determine attributes about the target. These attributes usually include the distance

from the laser source to the target and the target's shape, but other attributes such as reflectivity and similar optical features may be included. LiDAR can be implemented in many ways, ranging from static laser enclosures to rotating lasers to phased laser arrays. In addition, LiDAR can be applied along one dimension to find distance to a target, two dimensions to find distance to all objects at the same height as the laser, and three dimensions to obtain a complete rendering of a scene.

As with the other noted methods of course detection and position tracking, LiDAR has numerous benefits. One major benefit would be the fact that two- and three-dimensional implementations of LiDAR may readily be used to create mappings of a given area, as the discovered distance values at different points on a course can be matched to one another and overlaid to generate a map of a course. This relatively easy implementation of map-making then allows for advanced routing algorithms to be utilized, as the robot has a detailed knowledge of how its surroundings are positioned relative to itself. Another benefit of using LiDAR would be the fact that it can provide a high degree of detail. Of the commercially available LiDAR systems investigated, most were found to have an angular resolution of 0.5 degrees or smaller. This translates into LiDAR being able to detect narrow targets if necessary and being unlikely to fail to make note of small course features. Related to this would be the fact that the two-dimensional implementations of LiDAR investigated all possessed an effectively flat search space. In this sense, if the LiDAR systems are placed flat on a plane, they will scan along a plane, meaning that obstacles above or below the plane will not be detected. While this may seem to detract from LiDAR, in the case of this robot it is a benefit, as the only objects that could be considered obstacles are either the robot's objectives or fixed features close to walls. As such, it is not necessary for LiDAR to detect such features of the course. Finally, LiDAR benefits from the fact

that there is a multitude of open-source software available that implements the mapping of a region based on LiDAR readings. This would enable faster implementation of LiDAR, thereby allowing a greater amount of time to be spent on the development of controlling algorithms and improving the robot.

While LiDAR does possess many benefits, it does possess some negative features. The first and most notable negative would be the cost of LiDAR systems. Of the previously mentioned commercially available systems, the cheapest systems cost 38 dollars and only implement one-dimensional LiDAR. Systems implementing two-dimensional LiDAR cost at least 100 dollars, with many of them costing significantly more. Another negative feature would be the fact that two-dimensional LiDAR systems create a large volume of data compared to other course detection methods, typically in excess of 115 kbps. While this is a manageable volume of data for the Raspberry Pi being used, it must be noted that input and processing of such data will place a larger strain on the system compared to the less computationally intensive alternatives presented. A final downside of LiDAR is the fact that its performance can suffer in scenarios where there is significant disturbance in the air, such as in the case of weather like fog, rain, and snow. This occurs because the precipitation or water vapor present in the air interferes with the light that the LiDAR system produces in order to take its measurements. However, in the case of the robot, this concern is not applicable, as the robot will be directed to work in an interior space protected from the effects of weather.

Ultimately, the course detection and position tracking system chosen for implantation in the robot was a LiDAR system known as the RPLiDAR A1. LiDAR was chosen as opposed to the other potential methods for a few reasons. Firstly, it was chosen over camera-based feature recognition because while the two would not have differed in pricing to a significant degree, the



open-source software available, lesser data volume, and lower computational intensity of LiDAR made LiDAR a more appealing choice. In addition to that, it was determined that the ability to classify course features was not needed for the robot to complete its task. Secondly, LiDAR was favored over odometry due to odometry being prone to rapid error accumulation that would have likely led to catastrophic navigational failure on the robot's part. Odometry was not included in the robot as a complementary navigational system both to keep cost down and because research indicated that open-source LiDAR mapping software could provide sufficient accuracy by itself. Finally, LiDAR was selected instead of ultrasonic sensors because detailed mapping ability was desired in order to enable nuanced planning and routing with regards to locating and manipulating objectives.

The RPLiDAR A1 was selected over other commercially available LiDAR systems for three main reasons. The first reason was financially motivated, in that the RPLiDAR A1 was the cheapest two-dimensional LiDAR system found on the market. This helped ensure that the robot's cost will be kept as low as possible while still allowing it to make use of the benefits offered by a two-dimensional LiDAR system. The second reason would be that the RPLiDAR A1 was natively supported by open-source LiDAR mapping software that was compatible with the Raspberry Pi 3 Model B that was used to control the robot. This would enable the LiDAR to be implemented in the robot much more quickly than if such mapping software had to be developed independently, thus enabling the robot to be created faster and for more time to be dedicated towards improving aspects of the robot's performance. Finally, while other LiDAR systems, such as the RPLiDAR A2 and RPLiDAR A3, offered greater performance specifications for somewhat reasonable pricing, research revealed that the robot's operating

environment was of such a nature that such performance improvements would be unnecessary and ultimately result in money being wasted.

With regards to the software used in conjunction with the LiDAR, it was determined that the “hector\_mapping” ROS package would be used. Specifically, a derivative of that package, “RPLidar Hector SLAM” would be used in conjunction with the RPLiDAR A1 that had been chosen for the robot. This package would allow SLAM to be implemented in the robot and allow for such to occur without the need for odometry. This was notable because many implementations of SLAM rely upon odometry as an additional source of position tracking information. The “RPLidar Hector SLAM” package was chosen over other mapping packages because of the fact that it was specifically developed for use with the RPLiDAR series of 360 degree LiDARs, which would simplify the incorporation of mapping and position tracking into the robot.

### **c. Object Recognition**

To allow the robot to identify object color and position, some form of object recognition was needed. Object recognition required both some form of input sensor, such as a video camera, and some way to process that input and extract relevant data from it. To accomplish this, a Raspberry Pi Camera V2, which is a video camera, was selected as the input sensor. It was selected in favor of the numerous other video cameras on the market due to it being able to record 1080p30fps and 720p60fps video, while only costing 25 dollars. In addition to that, as indicated by the name, the Raspberry Pi Camera V2 had built-in compatibility with the Raspberry Pi series of microcontrollers, and as such would be able to be set up quickly. Additionally, there was a wealth of documentation available related to the setup of a Raspberry Pi Camera V2 on the

Raspberry Pi 3 Model B, ensuring easy setup. However, any camera with modern video recording formats would have been able to be used in the robot, and likely would not have had a notably more difficult setup process.

As previously noted, ROS was chosen as the development framework partially as a result of the numerous software packages available for it. Processing of the video from the Raspberry Pi Camera V2 was to be handled by one such package, referred to as “find\_object\_2d”. This package utilized a variety of computer vision algorithms specialized in identifying features from images, thereby allowing it to produce data related to the position of said features. In the case of the robot, these features would be the objects that the robot would want to identify and locate, such as the colored cubes and spheres that needed to be placed in corners. This package was chosen over similar packages because of the prevalence of documentation concerned with its setup and use, while simultaneously accomplishing the same goals of object recognition and determination of objects’ positions in a given video feed.

#### **d. Object Manipulation**

For the robot to score points by removing objects from the central 2-meter-wide circle in the course, some method was required to enable the robot to manipulate these objects. Any method chosen would have to adhere to the rules laid out by IEEE with regards to the physical specifications of the robot. Most notably, any implementation would have to fit within the maximum permitted size of the robot when it is in its starting square or in motion – 9 inches of length, 9 inches of width, and 11 inches of height. Additionally, said implementation would be permitted to extend further than 3 inches in length and width away from the robot whenever the robot was not in motion outside its starting square.

The first method of object manipulation considered was that of a collection area located at the front of the robot, bounded by the bumper required by the robot competition's rules. This area would likely be rectangular in shape, so as to ensure that any collected objects would be able to be easily packed alongside each other. Additionally, the area would need to be wide and deep enough to allow the collection of multiple objects at once while ensuring that objects would have difficulty accidentally removing themselves from the bin. Such a collection area would allow the robot to collect objects by simply driving into them, after which the depth of the bin and presence of its sides would mean that objects could only leave the bin if they rolled away from the robot. To deposit objects, the robot would only have to reverse away from them, as they would remain static upon the floor of the course. Such an implementation would benefit from being easy to include in the robot, given that it would only require that a space be set aside in the front of the robot for objects to be collected in. Additionally, this means that any sort of collection area scheme would cost practically nothing to implement. The first detriment to a collection area, however, consisted of having no method to ensure that objects do not escape the robot if they have a greater forward velocity than the robot. Second, such a collection area would require that the robot push its collected objects along the ground as it moves. This means that any motors selected for the robot would have to account for the opposing friction created by objects being moved. If this was not accounted for, the robot could be slowed to a standstill if it accumulated too many objects in its collection area.

A second method considered for object manipulation was that of an internal storage compartment running the length of the robot, positioned between the robot's wheels. This method bore many similarities to the previous method described, and as such also bore many of the same advantages and disadvantages. However, the exact implementation of such a length-

oriented storage compartment could mitigate or exacerbate these. Specifically speaking, the compartment could be up to 9 inches long, and would have to be at least 2.5 inches wide and tall, to ensure that both cubes and spheres could be stored within it. The compartment could have the floor of the course serve as its bottom, in which case it would have suffered the same friction-related issues of the previously described collection area. Alternatively, the compartment could have its own floor of some smooth material, thereby avoiding the issue of friction. However, this would have required the compartment to have a method of actively expelling its contents, as it would no longer be possible for the robot to simply reverse away from objects that it had moved. Furthermore, a compartment with its own floor would have required that the robot have some sort of active collecting device at the front of the compartment, as objects would need to be moved up from the course's floor to the collection area's floor. If the compartment were only 2.5 inches wide, then the front of the robot would likely have to be funnel-shaped to ensure that objects hit by the robot would be directed into the compartment, as otherwise the robot would need to be able to move precisely over objects to collect them. While this would not necessarily be impossible, such a need would introduce unnecessary complexity to the robot's design.

Ultimately, the advantages and disadvantages of the internal storage compartment were similar to those of the collection area. The storage compartment benefited relative to the collection area by potentially having larger storage space. Depending on implementation, it could have also benefited by not creating as much friction for the robot. However, the storage compartment was harmed by the fact that any additional advantage it had compared to the collection area would likely have required the creation of a collection device, thereby increasing development time, complexity, and cost. Additionally, the presence of a storage compartment in the center of the robot would have interfered with the placement of motors driving the robot's

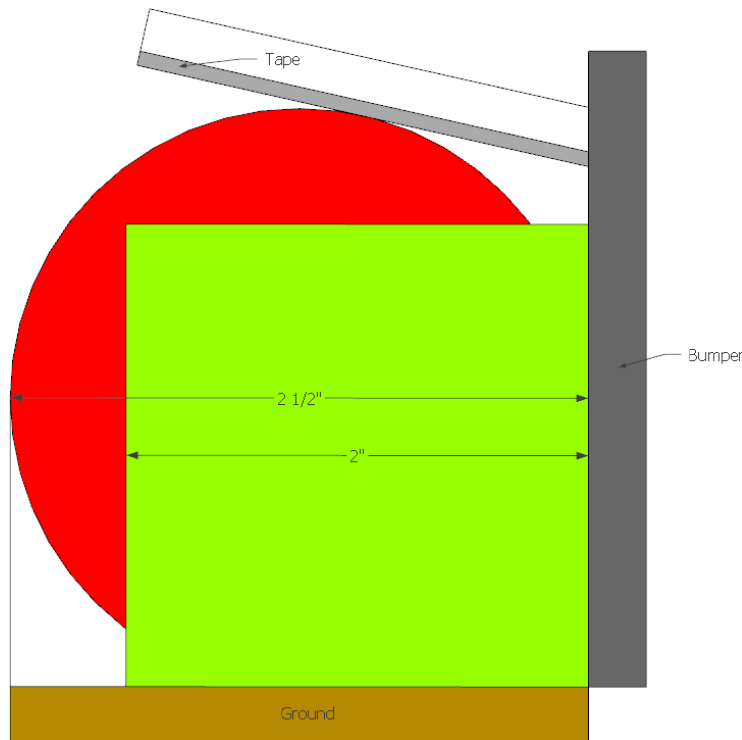
wheels, thus either limiting the size of a storage compartment or requiring the alternative wheel implementations be considered.

The final idea considered for object manipulation was that of a grabbing arm. Such an arm had been used in the 2018 robotics competition. However, the experiences associated with the creation and maintenance of the arm demonstrated that such a method of object manipulation was rather complex and prone to error. In short, these experiences consisted of having difficulty in design, testing, producing, and maintaining the arm. These problems all found their origin in the fact that a grabbing arm is a complex device that requires extensive planning and careful implementation, with any errors present in planning and implementation causing notable issues in the later operation of the arm. Additionally, the rules of the 2019 robotics competition dictated that any part of the robot may only travel 3 inches away from the robot in length and width, and such travel may only occur when the robot was not moving. These rules meant that any grabbing arm would need an entire object to be within 3 inches of the robot before it could interact with it. After that, the robot would have had to stop, and the arm would have had to manipulate the object into a collection area. As such, the robot would have only been able to collect one object at a time, and such collection would have taken much longer than the collection methods previously described for the collection area or storage compartment. Additionally, most implementations of a grabbing arm and its associated storage area would have needed to be placed on top of the robot, thereby interfering with the operation of the LiDAR used for course detection and position tracking. The only advantages conferred by the use of a grabbing arm would be its ability to be used to selectively collect objects, thereby enabling collection of one color of object at a time. This would have then allowed for objects to be deposited with maximum color-matching accuracy. However, the accomplishment of such would have required

precision tracking of object locations near the robot, as well as precision operation of the grabbing arm. The former was determined as being likely difficult to implement properly in the time frame available for robot design and creation, and experience from the previous year revealed that the latter was difficult to accomplish.

With these three methods of object manipulation in mind, it was easily determined that a grabbing arm would not be used. This was an easy decision to make, given that it presented many more disadvantages and challenges than advantages, and ensured greater simplicity in the robot's design. It was with the thought of simplicity in mind that the collection area was selected in favor of the internal storage compartment. While a storage compartment might have enabled somewhat superior manipulation of objects in the course, it was believed that the robot would be most benefited if any method of object manipulation were kept simple, given the difficulty caused in the previous year by a complex implementation of object manipulation. In the case of a simple implementation of the storage compartment, the advantages it offered were nearly identical to those offered by the collection area. The storage compartment also conferred additional disadvantages, such as potentially interfering with the placement of motors for the wheels. Additionally, a funnel shape would have had to have been placed at the front of the robot to ensure that precision movement of the robot over objects was not required. Given that a funnel shape would have taken some portion of the space that would have been required for a collection area, it was realized that a collection area would take up less space than the storage compartment, while conferring the same advantages and fewer disadvantages.

The design initially chosen for the collection area consisted of a rectangular area 140 mm wide and 30 mm deep located at the front of the robot, with its walls consisting of the bumper



*Figure 1. Side view of bar that prevents spheres from rolling forward.*

surrounding the robot. Placed within that area would be a thin rectangular bar, measuring 140 mm wide, 20 mm deep, and 5 mm high. It would be placed across the collection area 60 mm above the ground and would be angled 10° upwards. Placed on the underside of this bar would be DormTape™, with the sticky side facing downwards. This bar would then collide with the spheres being collected – albeit not the blocks due to their shorter height – and the adhesion of the tape would decrease the likelihood of the spheres being able to roll away from the collection area in the event of the robot’s velocity changing. The shape and function of this arrangement may be observed in Figure 1.

#### **e. Movement**

For the robot to be able to accomplish its goals, a movement system was needed for the robot. The movement subsystem consists of the motors that drive the robot, along with



whichever device or devices are used to transfer the power of the motors to the ground and induce movement in the robot. These devices come in many forms, and include wheels, tracks, and legs. For this robot, it was decided that wheels would be used. This decision lay in the fact that compared to tracks and legs, wheels are relatively simple to implement in a robot, as in their simplest implementation they need to be attached to a motor to drive them. Tracks, by comparison, require that a motor be attached to a driving gear, which must then be connected to at least one other gear by a system of connected tracks. Such an arrangement requires more space to be implemented, given that tracks must be spanned between the two gears, and involves a greater number of moving parts that may eventually break. While tracks would provide greater traction than most simplistic wheel implementations, the fact that the robotics competition's course consists of flat terrain means that exceptional traction is not needed to climb over terrain. Additionally, given that any submitted robots must include a low-lying bumper, objects may be prevented from getting underneath the robot, meaning that there need not be objects that the robot needs great traction to climb over. A system of legs was not chosen because such an arrangement would have been needlessly complex and difficult to implement without providing any notable advantages over tracks or wheels, given the course's level terrain. Additionally, it would likely prove difficult to make a system of legs provide the robot with more speed than that provided by either wheels or tracks. For these simple reasons, leg-based movement was not considered any further.

An arrangement of two drive wheels on the side and a roller ball bearing wheel was chosen for the robot. This arrangement of wheels was chosen because it avoided the necessity of the complex wheel control system associated with having four wheels like a car, wherein a system would be needed to turn either the two front wheels or two back wheels to permit the robot to

turn. Rather, with two drive wheels, turning may be accomplished by varying the amount of power received by each wheel. The robot would then turn in the direction of the wheel receiving less power. The ball bearing wheel in the back would serve to provide stability to the robot and ensure that the robot's back half would roll over the ground instead of dragging across it, thereby avoiding the creation of a large amount of friction. The two drive wheels chosen for the sides were Servocity 3.10" Press Fit Wheels, while the roller ball bearing wheel chosen for the back was a TruePower 10-9111 5/8" Roller Ball Transfer Bearing.

The Servocity 3.10" Press Fit Wheels were chosen because the robot did not require wheels that would be able to provide exceptional traction, given that the course consists of level terrain and the bumper around the robot ensures that the wheels will not collide with any objects. As such, the robot has no need for the traction required for climbing over objects or maneuvering over uneven terrain. The Servocity 3.10" Press Fit Wheels thus offered an acceptable degree of traction, while maintaining a simple design and the low price of 6 dollars per pair. The TruePower 10-9111 5/8" Roller Ball Transfer Bearing was chosen as the back ball bearing wheel primarily because of its low price, with 6 bearings costing a total of 13 dollars. These bearings were not needed for traction, given that they would not drive the robot through the course. As such, all that was needed of a ball bearing wheel was an acceptable price and decent construction. Additionally, these bearings were already in the possession of Murray State University's IEEE group, thereby further saving costs.

With regards to the motors needed for the robot, it was realized that the flat terrain of the course and requirement of a bumper encircling the robot meant that the motors did not need to provide large amounts of torque. This was because the presence of flat terrain ensured that any

torque produced would be able to be efficiently used to power the lateral movement of the robot across the course. This would have been opposed to if hilly terrain had been present, in which case some amount of produced torque would have been expended moving the robot vertically rather than horizontally. At the same time, the requirement of a bumper ensured that objects would not get lodged underneath the robot, provided that the bumper extend most of the way to the ground. This meant that the robot's motors would not need to provide the torque required for the robot to climb over obstacles, as there would be no obstacles for the robot to climb over. With these two elements in mind, it was realized that motors would only have to provide enough torque to move the robot and a reasonable quantity of objects that needed to be pushed. This then meant that motors could be chosen with speed in mind, provided that they also remain compatible with the Arduino Uno microcontroller that would be used to control them.

Three motors compatible with the Arduino Uno were considered for initial inclusion in the robot. These were the Parallax Feedback, Parallax Continuous Rotation, and Hitec D485HW motors. The motor specifications considered were maximum rotational velocity, operating voltage, current draw, torque, and price. A comparison of these specifications may be seen in Table 2.

Table 2. Comparison of Motors

Motor	Max. Rotational Velocity	Operating Voltage	Current Draw (no-load)	Maximum Torque	Price per motor
Parallax Feedback	120 RPM	5-8.4 VDC	140 +/- 50 mA	34.7 oz-in	\$27.99
Parallax Continuous Rotation	50 RPM	4-6 VDC	140 +/- 50 mA	38 oz-in	\$14.99
Hitec D485HW	66.6 RPM	4.8-7.2V	320 mA	104 oz-in	\$44.99

From the table, it may be seen that the Parallax Feedback motor would provide the greatest maximum rotational velocity with 120 RPM. When compared to the Parallax Continuous Rotation motor, it would do so while having the same current draw and nearly identical torque. However, the Parallax Feedback motor would have a slightly greater cost per motor. Compared to the Hitec D485HW, the Parallax Feedback motor would provide nearly double the maximum rotational velocity while drawing approximately half the current, and both would be done at a lower price per motor. However, it would only provide a third of the maximum possible torque. Given the prioritization of motor speed over torque, it was decided that Parallax Feedback motors would be used in the robot, despite their greater cost when compared to the Parallax Continuous Rotation motors. The Hitec motors were not chosen due to their greater cost and current draw, and lack of comparable rotational speed.

#### f. Power

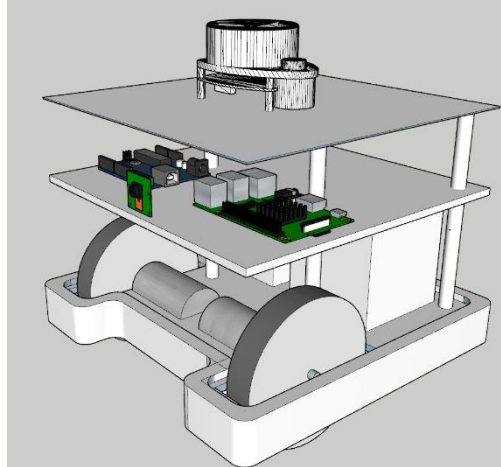
Many options were available for the robot's power storage, given the vast quantity of unique batteries that exist in the world. As such, a battery was selected based on the relevant constraints

on its parameters. Namely, the battery needed to have sufficient capacity given a voltage of 5 V, as was intended to be needed to power all connected devices, and an estimated current draw that would allow the robot to operate for at least 3 minutes. The current draw was estimated by the method described on the Powerstream Technology website [5], although an assumption of maximum current load is used instead of calculating a potential average current draw. This was done because numerous electronic systems are present in the robot, with some of them possibly drawing a wide range of power. Thus, it is better to overestimate the battery capacity needed rather than to underestimate it. The calculation can be seen in Appendix D, and the minimum battery capacity necessary was found to be 6.5 Ah. This minimum battery capacity was calculated with the idea of the robot being able to operate several times in a row without charging.

In addition to these specifications, a fact of note was that in the previous year's submission by Murray State University IEEE group to the robotics competition, 12 V batteries were used to power 5 V devices. This was accomplished without burning out the 5 V devices by sending the current from the 12 V batteries through a Raspberry Pi motor shield that stepped down voltage to 5V. With this, the robot was to incorporate the 3 Tenergy 2 Ah Nickel-Metal Hydride batteries that were used in the previous year's IEEE robot. Together these 3 batteries would provide 6 Ah at 12 V, which ideally translated to 14.4 Ah at 5 V. This was more than enough battery capacity for the robot and ensured that the robot will be able to continue to operate even if one or more batteries disconnect during its operation. As such, it was not necessary to seek out a different battery for the robot.

g. **Physical Structure**

The initial structure of the robot, as depicted in Figure 2 and Figure 10 in Appendix B, was selected based on a number of desirable characteristics. The most desired characteristic of the robot's structure was to adhere to the physical constraints provided by IEEE. These constraints are featured in Appendix A. Failure to adhere to these constraints would cause the robot to be automatically disqualified from the robotics competition. Another desirable characteristic was that of having enough space to make the maintenance and repair of the robot as easy as possible. The desirability of this characteristic came from experience garnered during the construction and testing of the IEEE robot produced last year, wherein it was found that having little space between layers of the robot and elements on those layers lead to multiple instances of difficult maintenance. Maintenance was made difficult primarily because the lack of space hindered attempts to reposition wires and connectors without needing to take apart the entire robot. In addition to that, the cramped conditions within the robot caused the repositioning of components and connectors to move nearby components and connectors, which on several occasions led to those moved components and connectors needing to be repositioned to ensure the robot's proper operation.



*Figure 2. Front view of robot*

A third desirable characteristic was that of having enough space around the computational components of the robot, namely the Raspberry Pi, to allow them to passively cool without hindrance. This trait was desirable because initial testing of the Raspberry Pi demonstrated that it could heat up relatively rapidly when under heavy load, such as that caused by computer vision processes. If airflow was not able to reach the Raspberry Pi and aid in its passive cooling, then there was the risk of the Raspberry Pi heating itself to the point of decreased computational efficiency or emergency shutdown. In the former case, it is commonly known that electronic devices such as CPUs and GPUs perform worse when permitted to heat up, with performance decreasing more as temperature increases. In the latter case, those who work with or read about CPUs and GPUs often know that such devices will typically shut down when they reach 100°C to prevent them from suffering permanent damage due to extreme temperatures.

Another desirable characteristic was that of ensuring that the RPLiDAR A1 had clearance in all directions, as a failure to do so could result in the detection of the robot's internal features rather than the external course's features. This would have the effect of hampering the mapping of the course with the LiDAR, as features hidden by the robot's internal features would not be

able to be mapped without the robot's rotation and movement around the course. In addition, this could have led to the robot improperly believing that it was always located near a wall and that it ought to move away from the wall. While neither effect would have been insurmountable, both were undesirable, and the structure reflected this. The final desirable characteristic of the robot's structure was that of low cost. The benefits of a low-cost structure are that damaged structure components could be replaced at little to no cost and that the robot would be cheaper to produce. The cheap replacement of damaged structure components was desirable because it ensured that the budget wouldn't increase simply because some part of the robot's structure broke.

With these characteristics in mind, an open, layered structure was selected for the core of the robot. This structure was selected because it ensured that each layer had enough space for maintenance and repair to take place without incident, allowed air to easily flow across the Raspberry Pi, and fit within the constraints imposed by IEEE. This structure was chosen over a closed, layered structure because the latter would have worse airflow around the Raspberry Pi and require more work to repair, as any panels or box covering the robot would have to be removed to access the electronics of the robot. In addition to that, such coverings on the robot would have simply been prone to damage if the robot were to run into any obstructions that were not first encountered by the IEEE-mandated bumper, which would then increase the robot's maintenance cost. Finally, the initial creation of robot body coverings would incur an unnecessary financial cost, given that they present no notable benefit to the robot's operation. One could argue that such covering would prevent the robot's electronics from experiencing damage if a foreign object entered the robot's body. While this is true, under the dictated operating conditions of the robot, the robot should not encounter any objects capable of entering its body and damaging its internals.



There were three layers present, with the first layer holding the motors, wheels, batteries, and bumper. The second layer held the Raspberry Pi computer, the Arduino microcontroller, and the Raspberry Pi Camera. The third layer only served to hold the RPLiDAR A1. The three layers were connected to one another by means of threaded metal dowels or rods extending through the entire structure of the robot, with washers and nuts securing each layer at the proper elevation. Each layer would have a height of 10 mm and edge lengths of 210 mm, except for the first layer. While the first layer had matching outer dimensions, a slot 140 mm wide and 30 mm deep was to be centrally cut into its front to serve as the object collection area. In addition to that, two 30 mm wide and 110 mm long slots were cut into each front side to accommodate the robot's wheels. The first and second layer were separated by 70 mm, as were the second and third layer. The threaded rods supporting the layers were 12.7 mm in width and 170 mm in length. The layers were made from acrylic sheets, with holes cut by laser to serve as attachment points for components. A 3D printed bumper 10 mm in width with radii of curvature of 10 mm would surround the first layer of the robot to adhere to IEEE-determined constraints, and would sit 5 mm above the ground, having a height of 70 mm. This bumper would be printed from nGen 3D printing filament, which is made from Eastman Amphora™ AM3300 3D polymer. This filament was selected over other similarly priced filaments because of its superior strength, quick solidifying, and lower print temperature.

## **V. CHANGES TO SUBSYSTEM AND FUNCTIONAL ANALYSIS**

### **a. Control**

After creation of a robot prototype began, it was soon realized that the choice to use the ROS to ease development had been based on the assumption of the ROS being able to be installed

without issue, and that the ROS would then proceed to work correctly on the Raspbian operating system that had been installed. Over the course of the first month of development, attempts were made to use a multitude of video and text-based tutorials for the installation of the ROS.

However, these attempts suffered from a variety of issues. These included files being installed to incorrect locations to the activation of ROS packages taking incredibly long times. While the former issue was able to be resolved after extensive troubleshooting and research on a variety of robotics-oriented internet forums, research revealed that the latter issue stemmed from the fact that the Raspberry Pi 3 Model B microcontroller only had a processor frequency of 1.2 GHz and 1 GB of RAM. Specifically, for any form of work to be performed with the ROS, the ROS had to be “built” after any of the packages for the ROS had been installed or changed. Research discovered that this process is known to take a long period of time, even on computers with notably better processors and greater quantities of RAM compared to the Raspberry Pi 3 Model B. As such, any attempt to build the ROS on the Raspberry Pi was certain to take a long period of time. While attempts were made to minimize the number of times that the ROS needed to be “built,” this problem continued to be a notable issue, and ultimately contributed greatly to the eventual decision to abandon the ROS.

After the decision had been made to abandon the ROS, it was decided that further development of the robot’s software would be accomplished using the coding language known as Python. Python was chosen over other coding languages because Python had been previously used for research and other school projects. While this shift to Python meant that ready access to open-source packages sharing a common framework – that of the ROS – was lost, it also meant that greater control over the development of the robot was enabled. This would be permitted by knowing how nearly every element of the robot’s control was coded, and thus there would be

few to no scenarios in which a software package would have the opportunity to act as a black box wherein its underlying functioning would be unknown.

It should be noted that the shift away from the ROS meant that course detection and position tracking would have to be manually developed, and this will be discussed within section b. The same change also happened within the scope of object recognition, which had impacts that will be discussed within section c. However, it must be noted that the changes made to the object recognition processes had an impact on the control of the robot. Namely, they necessitated the usage of a second Raspberry Pi 3 Model B, which was based on the need for computational power that could not be supplied by the main controlling Raspberry Pi. These two Raspberry Pi 3s communicated data with one another by means of serial connections, although the situation was ultimately that the additional Raspberry Pi only communicated its processed data to the main controlling Raspberry Pi, and not the other way around.

Another change that occurred related to the microcontrollers used was the abandonment of the Arduino Uno. This decision was made due to developments in the power distribution system of the robot. These developments meant that the Arduino Uno was no longer needed for its ability to distribute power from the Raspberry Pi to the motors, as such distribution was instead handled by a dedicated power regulator and motor shield. Given that the Arduino Uno was included primarily for its ability to safely distribute power to the motors, this meant that the Arduino Uno no longer had any specific use and could be safely abandoned. The specifics of the power distribution development will be discussed within section f.

## **b. Course Detection and Position Tracking**

As previously discussed, plans for the development of the robot initially called for the combined usage of the RPLiDAR A1 and the “RPLidar Hector SLAM” mapping package for the ROS. Attempts to implement and use the package were made while the ROS was still being used. However, such attempts were also plagued by numerous technical issues, albeit these ended up being able to be solved. The most notable issue that came to bear was that of the data being collected by the LiDAR not being provided to the “RPLidar Hector SLAM” package, thereby preventing it from generating a map of the course. This issue was eventually resolved when research revealed that several lines in a specific file determined the location to which the LiDAR’s data was being sent. Several variations of data targets had to be tested before the right combination was found, after which the “RPLidar Hector SLAM” package produced a map of the course. However, even though success was had in generating a map, the position tracking and map updating abilities of the package were partially successful at best. In this sense, testing revealed that even slow movement of the LiDAR within the boundaries of the course resulted in inaccurate position reporting by the package. In addition to that, as the package continued operating, the updating of the generated map with additional data from the LiDAR would overlap previously generated sections of the map. This meant that over time, the map would accumulate a large number of erroneous course geometry depictions, that would likely hamper the robot’s ability to use the map to generate a path to follow.

However, while the generation of a map had been achieved, research revealed that there was practically no documentation showing how a generated map could be used by a robot to move through an area. Additionally, the ROS packages that were found that could be used for path

determination were poorly documented, meaning that it was nearly impossible to determine the proper usage of any functions provided by the packages. The dearth of documentation thus proved to be a roadblock to the further development of course detection and position tracking, as well as its confluence with the robot's movement. This contributed to the decision to abandon the ROS in favor of Python.

Once Python had been adopted for robot development, course detection and position tracking was implemented relatively simply. To process data provided by the LiDAR, a function was written that would collect the data produced by the LiDAR and place it in a two-dimensional array. The first column of the array stored the distance values recorded by the LiDAR, which were reported in millimeters. The second column of the array stored the angle measurements associated with the distances measured, which were reported in degrees. This array would then be output whenever the function was referenced by another function and could be manipulated to accomplish a variety of goals. The exact manner of how this was done, as well as how it contributed to the robot's ability to function as a whole, will be described in section VI.

### **c. Object Recognition**

Compared to the other elements of the ROS previously discussed, the “find\_object\_2d” package proved relatively simple to implement in the robot. The only major hurdle that appeared was some difficulty in having the video feed from the camera be sent to the proper location in order to enable the “find\_object\_2d” package to use it. However, this was resolved quickly with some basic research, as similar issues were well documented on the internet, and many users of the “find\_object\_2d” package had already solved the problem and provided their solutions to the greater ROS community.

With the “find\_object\_2d” package working properly, work began on getting it to identify the cubes and spheres that would be present on the course. While this task did not prove difficult, a roadblock quickly became apparent in the fact that the “find\_object\_2d” package consumed a large amount of system resources whenever it was running on the Raspberry Pi 3. This meant that any other processes had difficulty in getting sufficient resources allocated to them, and as such would operate at a much-reduced speed. For example, when “find\_object\_2d” was run in conjunction with “RPLidar Hector SLAM”, the map produced by the latter would only update after several seconds. This was opposed to its standard map-updating-speed, wherein it would update the map several times in a second. While this issue may have been resolved by running the “find\_object\_2d” package on a separate instance of the ROS on a second Raspberry Pi microcontroller, the other issues experienced with the ROS in this time frame lead to the abandonment of the ROS. This was in opposition to trying to use the ROS in conjunction with multiple microcontrollers.

After the move to Python occurred, work on object recognition was halted for a time in favor of working on the course detection, position tracking, and movement subsystems. Work resumed approximately halfway through the spring semester. When this occurred, it was determined at the time that a method of detecting objects had to be devised that did not rely as heavily on the usage of computer vision. This determination was made with the previous experience of excessive computational resources being used by “find\_object\_2d” in mind, and lead to the attempted use of the TCS3200 color sensor for color recognition. This decision was made because several TCS3200 color sensors were already present in the IEEE lab, and they had been used in previous years’ robots for color detection. In the arrangement devised, several of these color sensors would be placed such that they could detect the color of objects collected by the robot in its

collection area as it moved about the course. At the same time, a color sensor would be installed on the left side of the robot, such that whenever it moved counterclockwise around the central structure it would be able to detect the color of whichever side of the central structure was facing it. With these two sets of color sensors in place, the most prevalent color detected by the sensors would be compared to the central structure's color. Then, actions would be taken depending on whether the colors matched.

This plan soon demonstrated that it would not function. The TCS3200 color sensor was tested, and this revealed that such color sensors were inaccurate in their reporting of color. The TCS3200 color sensor functions by exporting 3 values, which correspond to the intensity of the colors red, green, and blue that it detects. The color sensors were tested by connecting one color sensor to an Arduino Uno microcontroller, after which a section of open-source code provided by the supplier of the color sensor was used to extract data from the color sensor. Initial testing was conducted with an uncalibrated color sensor in standard lighting conditions. When red cubes and spheres were placed in front of the color sensor, it reported that the majority color present was green. Similar results were produced when green and blue objects cubes and spheres placed in front on the sensor. After this, the sensor was calibrated by recording each set of values produced by the color sensor when a black object and white object were set an inch in front of the color sensor. All values produced by the color sensor were then scaled from 0 to 255, with 0 referring to the lack of a specific color and 255 referring to the maximum presence of a color. After this calibration occurred, initial testing was conducted again under standard lighting conditions by simply placing a variety of colored cubes and spheres in front of the color sensor. However, once again, the color sensor demonstrated inaccuracy by failing to report a majority of red, green, or blue when respectively colored objects were placed in front of it. At this point,

attempts were made to place a shroud around the color sensor – to prevent outside light from influencing its readings – and use the LEDs attached to the color sensor to improve results. However, once again, the color sensor failed to report that objects of red, green, or blue color had a majority of those colors. This was despite the fact that those were the colors that it specifically detected, and all efforts had been made to calibrate it and reduce the influence of outside light on its readings. When this was combined with other work that had been pursued in parallel, it was decided that the TCS3200 color sensor would not be used.

Specifically, progress had been made in the use of two separate cameras – a Raspberry Pi Camera V2 and a Logitech C270 web camera – in conjunction with a Python library called OpenCV to conduct color recognition and comparison. The Raspberry Pi Camera V2 had a wide-angle lens attached to it and was oriented to look into the robot’s object collection area. The Logitech C270 web camera, which cost 20 dollars and was capable of 720p24fps video capture, was oriented to look left, such that it would be able to record video of the central structure whenever the robot was orbiting counterclockwise. OpenCV was composed of numerous functions related to computer vision. While this choice would seem to have gone against the decision to not use computer vision, the OpenCV functions were only used to conduct simple image processing and color extraction on the video feeds provided by the two cameras. This would be opposed to the computer vision done by the “find\_object\_2d” package previously used, wherein a variety of more computationally intensive functions were being used to analyze a video feed and determine if any frame of the video contained an object that may be arbitrarily oriented and positioned. In this new implementation of color recognition, the video feeds were taken in by the Raspberry Pi microcontroller. They were then cropped so the only areas analyzed were those of interest, such as the regions showcasing the object collection area and central



structure. After the videos were cropped, image processing was conducted on the video feeds to identify and ignore any black, grey, or white colors that may be present within the videos, as these colors would be present in the course and detection of these colors was not desired. This was accomplished by experimentally determining the thresholds of hue, saturation, and lightness associated with these colors, after which any colors that exceeded these thresholds were ignored by the code. Image processing was then conducted to detect the red, green, and blue values present within the cropped video feeds. From these values it was possible to determine whether a video feed was showing a majority of the red, green, blue, or yellow colors, all of which may be present by means of being the colors of the objects that needed to be collected.

This method of color recognition proved successful and could reliably identify the color of objects placed in front of the cameras. It functioned most successfully when exposed to individual colors with a good light source present. Its success rate would decrease as lighting got worse, or as multiple colors were placed in front of it. The easiest combinations of colors to differentiate were those of red and blue and red and green, with the reason for this being that these colors as manifested in the competition were relatively distinct. The hardest combinations of colors to differentiate were red and yellow, blue and green, and yellow and green. This may be explained by the fact that the colors present on the course objects were not very intense, and that these colors are somewhat closely placed to one another on HSV color charts and the electromagnetic spectrum. The latter reveals that those sets of colors are relatively similar in their visual nature, meaning that it would have been rather difficult to achieve distinct recognition of these color combinations without potentially excessive testing. Additionally, such testing and improvement of color recognition would have likely been wasteful, as the lighting conditions that would be present for the robotics competition were not known. Thus, any such

intensive testing may have easily been neutralized by the robotics competition having notably different lighting conditions compared to test lighting conditions. As such, inaccuracy in the recognition of color combinations was considered acceptable.

Furthermore, inaccuracies in color combination recognition were acceptable because of how the color recognition would be used. As noted earlier, OpenCV was being used to allow color recognition and comparison. The latter was accomplished rather simply, in that once color recognition had occurred and majority colors were being reported, the color that each camera feed reported was compared. If the colors were the same, the robot would know that the majority color of the objects that it had collected in its movement matched the color present on the side of the central structure facing the robot, and thereby the color of the quadrant that the robot was in. This knowledge would then allow the robot to conduct certain actions, which will be further described in section VI

#### **d. Object Manipulation**

Of all the subsystems implemented in the robot, the object manipulation subsystem underwent the least change during development. This was primarily because it was not implemented until the later stages of development, after the robot was capable of basic navigation around the course. In its first implementation, the object collection area was created as previously described, although the creation of the width-spanning bar with tape was delayed. This was done in order to simplify initial prototype creation, as the bar was the most complex element of the collection area to implement. Additionally, delaying the bar's implementation allowed observation of what object manipulation performance was able to be achieved without the bar. Initial testing revealed that the collection area had perfect performance with regards to

collection of cubes, which was expected given their inability to roll away. The spheres proved more difficult to collect, but the low speed of the robot prototype meant that they seldom rolled away. Such occurred only when the robot attempted to perform a zero-point turn, wherein the spheres would sometimes manage to escape beyond the sides of the collection area. However, this was primarily only an issue when other spheres or cubes were present, as the interaction of their geometries and the rotational movement of the collection area during a zero-point turn allowed spheres to be pushed forwards by other objects. After the spheres were pushed out of the collection area, they were no longer bounded by the collection area's geometry, and thus the collection area would complete the rest of the turn with the robot and without the sphere that was pushed out.

To mitigate the issue of spheres rolling out of the collection area and enable the collection of more objects, the collection area was improved by increasing its width and depth. This resulted in a collection area 188 mm wide and 56 mm deep. Inclusion of the tape bar was further delayed in favor of further testing without the bar and being able to use the time that would be required to create the bar on other pursuits. Testing revealed that the greater depth of the new collection area greatly mitigated the issue of spheres being pushed out by other objects, although it was not fully resolved and would remain an occasional issue throughout the robot's lifespan. More rigorous testing conducted at this time showed that there was another issue with the collection area, though. This issue was that too many cubes could be collected by the robot, and the inability of cubes to roll meant that their combined kinetic frictions could prevent the robot from moving forward. The issue was resolved by the installation of motors with more torque and lower maximum rotational speed, which will be further detailed in section e. However, the installation of said motors resulted in more spheres rolling away from the collection area whenever the robot

slowed down. While this appears counterintuitive given the motor's lower maximum rotational speed, it can be explained by the fact that the lower-torque motors were incapable of operating at their maximum rotational speed, and thus the robot never moved quickly enough for sphere-rolling to be a problem. The higher torque motors, on the other hand, were more easily able to attain their maximum rotational velocity, meaning that the robot moved faster. Thus, whenever the robot might stop to turn, the spheres would retain this speed and roll away from the collection area.

While it may seem that the solution to this problem would have been to create and install the tape bar that has previously been described, the concurrent development of the 2-camera color recognition system prevented this. This was because the camera looking into the collection area needed an unobstructed view of the objects within the area, and the tape bar would have had to be positioned such that it would have blocked the majority of the camera's view of the objects. Additionally, the arrangement of motors and electronic components within the robot meant that the camera could not be moved lower to see under the bar, and the fact that the bumper had to be the outermost portion of the robot whenever the robot was in motion meant that the camera could not be placed above and in front of the bar. As such, to allow color recognition and comparison to be conducted and thereby enable the collection of points related to objects being placed in the correct color corner, the removal of the tape bar was considered necessary and the possibility of spheres rolling away was acceptable. This acceptance of sphere-rolling was partially influenced by the fact that when the robot was further away from the center, spheres that rolled away would occasionally have enough velocity to remove themselves from the central area. Additionally, when the robot was closer to the center of the course, spheres that rolled away would usually be collected by the robot later in its operation.

#### **e. Movement**

The main change that was made to the robot's movement during implementation of subsystems was the change from Parallax Feedback motors to the Servocity 101 RPM Gear Motor. As mentioned in the previous section, this change occurred after it was seen that collection of too many objects could cause the robot's motors to stall due to lacking the torque needed to propel the robot. The Servocity 101 RPM Gear Motor was selected because it could provide 57 oz-in of torque – approximately 64.2% more torque – while only having 15.8% less maximum rotational speed compared to the Parallax Feedback motor. Additionally, the Servocity 101 RPM Gear Motor could be purchased for 24.99 dollars per motor, which was 3 dollars cheaper.

With regards to the wheels, the main change that occurred was the 3D printing of a wheel body compatible with the driveshaft of the Servocity 101 RPM Gear Motor. The tires of the 3.10" Servocity Press Fit wheels was retained, being removed from the original bodies of the wheels and placed on the 3D printed body. A development to the wheels that almost occurred was abandoning the usage of the rubber tires from the Servocity Press Fit wheels and instead using a 3D printed wheel with spiked teeth, similar to how a car tire may have studs for gripping icy roads. This idea was conceived when testing was being done the day before the robotics competition in response to the very occasional slippage of the wheels on the course that would be seen whenever the robot collected many objects. Given that the 3D printer was not being used at the time, an attempt was made to manufacture the wheels. However, the ability of the 3D printer to do detailed printing had been overestimated, and as such the wheels produced had notable texture without having spikes large enough to grip the carpet of the course. The wheels were



*Figure 3. Angled view of attempted spiked wheel. Hot glue was placed to try to increase traction, but ultimately decreased it.*

made from the same nGen 3D printing filament used to print the robot's bumper. While the lack of grip may have been caused by the 3D filament used, it was not possible to attempt to print the wheels from another filament type, as the only filament present was the nGen filament. An attempt to reprint the wheels with larger, more distinct spikes was considered, but was not done because of a need to print other, more vital components. The wheels that were produced may be seen in Figure 3.

#### **f. Power**

The power subsystem underwent a variety of changes during the robot's development. The main change was the shift from 3 Tenergy 2 Ah Nickel-Metal Hydride batteries to 3 2Ah Lithium Ion batteries, with each supplying 3.7V. While this would seem insufficient for producing the 5V the robot needed, placing these batteries in series allowed for a voltage of 11.1V to be achieved. This voltage could then be stepped down to 5V. This change was made

because of the fact that the lithium ion batteries took up much less space and weighed significantly less than the previous arrangement of 3 Tenergy batteries. Specifically, the 3 Tenergy batteries had a volume of  $359.736 \text{ cm}^3$ , while the 3 lithium ion batteries had a volume of  $56.8673 \text{ cm}^3$ . This enabled a great deal of space to be saved while maintaining roughly the same voltage supplied to the robot. However, what had been forgotten was the fact that placing the lithium ion batteries in series to increase their voltage meant that they had a combined capacity of 2 Ah, rather than the 6 Ah that would have been achieved if they had been placed in parallel. Additionally, the nature of lithium ion batteries meant that as they were drained of power by the robot, their voltage would drop. As the voltage dropped, more current would be drawn. As such, while the batteries had a capacity of 2 Ah, they lasted for a much shorter time than a single Tenergy battery. The result of these factors was that the lithium ion batteries would be expended within the course of a few hours of testing. Furthermore, the charging setup that was able to be created was not able to charge the batteries very quickly. Given that none of the robot's developers were well versed in electric power supply, a superior solution could not be devised. As such, while the batteries would be expended within a few hours, they would take in excess of a day to charge.

When this problem was experienced, it was realized that returning to the Tenergy Nickel-Metal Hydride batteries was necessary. However, if 3 batteries were used again, the same problems caused by the batteries' weight and size would be known. The idea was forwarded to try using only 1 battery, given that the nickel-metal hydride battery ought not to suffer from the same degree of voltage drop as the lithium ion batteries. The arrangement was tested and was found to work well. While the single Tenergy battery did only have  $1/3$  the energy capacity of the previous Tenergy battery parallel arrangement, it provided power for longer than the

arrangement of 3 lithium ion batteries. Additionally, 5 Tenenergy batteries were available, meaning that batteries could be charged ahead of time and replaced as they discharged. As such, the final arrangement of batteries consisted of 1 Tenenergy 2 Ah Nickel-Metal Hydride battery.

The second and third major changes that occurred to the robot's power system was the shift from using an Arduino microcontroller to supply power to the motors to using a motor shield and the implementation of an independent power regulation system. The former was implemented to take in 12V power from the batteries and distribute it to the motors, which were controlled by a connection between the main controlling Raspberry Pi microcontroller and the motor shield, and feed power to the power regulator. The power regulator would step the voltage down to 5V and feed it to the Raspberry Pi microcontrollers, while ensuring that the microcontrollers and any of their connected devices, such as the LiDAR and cameras, would not be exposed to excessive voltage or current in the event of battery failure or other events. Together, these changes meant that the robot was mostly protected from catastrophic damage, although a short circuit created between electronic components could still ruin a microcontroller or device.

#### **g. Physical Structure**

The structure of the robot changed significantly over the course of development, going from the previously described 3-tier acrylic structure with a 3D printed bumper to a much shorter 3D printed structure and bumper arrangement with "1.5" tiers. This occurred because the prototypes of the robot structure were built with the priority being to have a somewhat correct robot body that could be used as a testbed for other subsystems. Later versions of the robot were made after a reliance on 3D printed components had emerged and with an ultimate goal in mind: trying to lower the robot enough that the RPLiDAR A1 could be used to detect the space hotels positioned

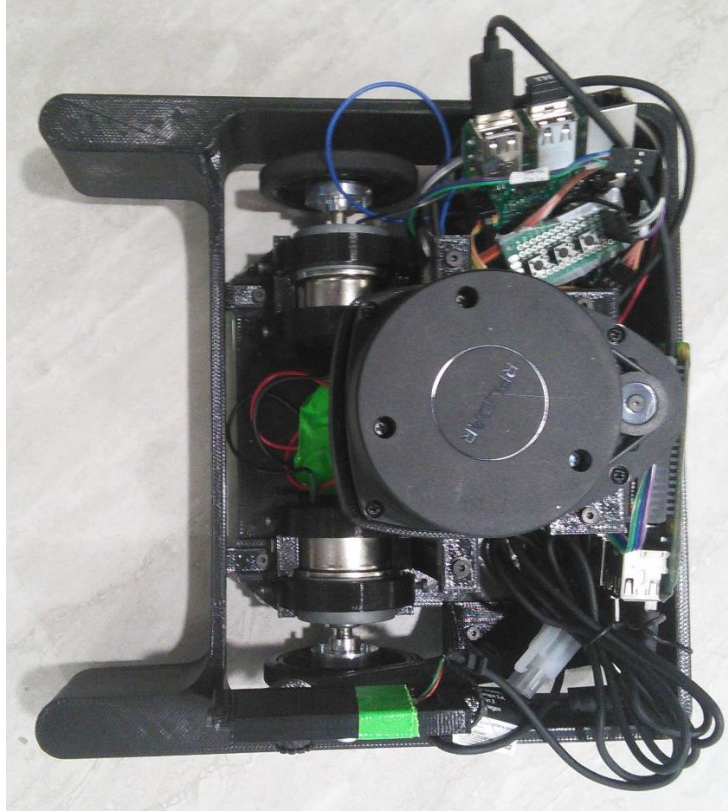


around the central circle. With regards to the prototypes, the fact of the matter was that the it seemed more important to get the various subsystems of the robot working than to create the body of the robot in complete accord with the original plan. With regards to the later and final versions of the robot, the second revision of the robot competition's rules was incorrectly interpreted. Within these rules, a piece of vague wording could be interpreted to mean that the space hotels were placed on blocks 3 inches tall, rather than on blocks that were 1 inch tall. When this was combined with the fact that the space hotels were 1.8 inches tall, it was believed that it would be possible to decrease the total height of the robot to 4.8 inches. If this had been achieved and the interpretation of the wording had been correct, the RPLiDAR A1 would have been able to detect the space hotels, which would have enabled the robot to evade these structures.

In response to the incorrect interpretation and desire to make a shorter robot, a robot body that allowed the RPLiDAR A1 to sit below 4.8 inches of height was created by 3D printing. This limited height necessitated a very compact arrangement of internal components to the robot and made working on the robot difficult. In addition to that, a newly published version of the robotics competition rules was released by IEEE that provided a schematic of the course, demonstrating that the space hotels would have a total height of 2.8 inches, rather than 4.8 inches. This meant that it was now impossible for the robot to be designed such that the RPLiDAR A1 could detect the space hotels, as it would have to sit at the same height as the wheels to do so. However, the design was not changed significantly after this was known. This may be explained by the fact it was felt that other tasks took priority to the creation of an entirely new robot body, such as the creation of necessary 3D printed components to hold various devices in the robot. In addition to

that, the fact that returning to the original design or some variant of it would have made working on the robot much easier over the course of the semester was not considered.

As such, the elements of the robot body that changed after the creation of the low-lying robot were the motor mounts, the LiDAR mount, the camera mounts, and the bumper. The motor mounts were changed because the Servocity 101 RPM Gear Motors were larger than the original Parallax Feedback motor and were only changed once. The LiDAR mount was changed to increase the height of the mount, and thus increase the amount of space available for internal components and working. This increase in height occurred several times, and ultimately resulted in the top of the LiDAR sitting 6 inches above the ground. The camera mounts were not included in the original robot design, and thus had to be added in. One mount was attached to the underside of the LiDAR mount, and permitted the Raspberry Pi Camera V2 that it held to swivel vertically and view the collection area. The other mount attached to the back left of the robot's body and held the Logitech C270 web camera above the center of the robot's left side. This permitted the camera to view the central structure of the course. The final construction of the robot may be observed in Figure 4 with a front-facing view of the robot being found in Figure 11 in Appendix B.



*Figure 4. Top view of final robot.*

## **VI. CONTROL LOGIC**

For the robot to operate, software must exist that can take in processed sensor data and use it to control the robot. This exists separate from the control subsystem, which primarily encompasses the hardware and middleware that controlling software runs on. Prior to the beginning of robot's implementation, it had been planned to use the ROS and a variety of packages to simplify the process of controlling the robot. However, as previously noted, this development route was abandoned after the ROS proved too cumbersome to use. This section will address the overall control logic that was developed within the coding language Python, outside of the scope of code directly related to the functioning of individual subsystems. Before

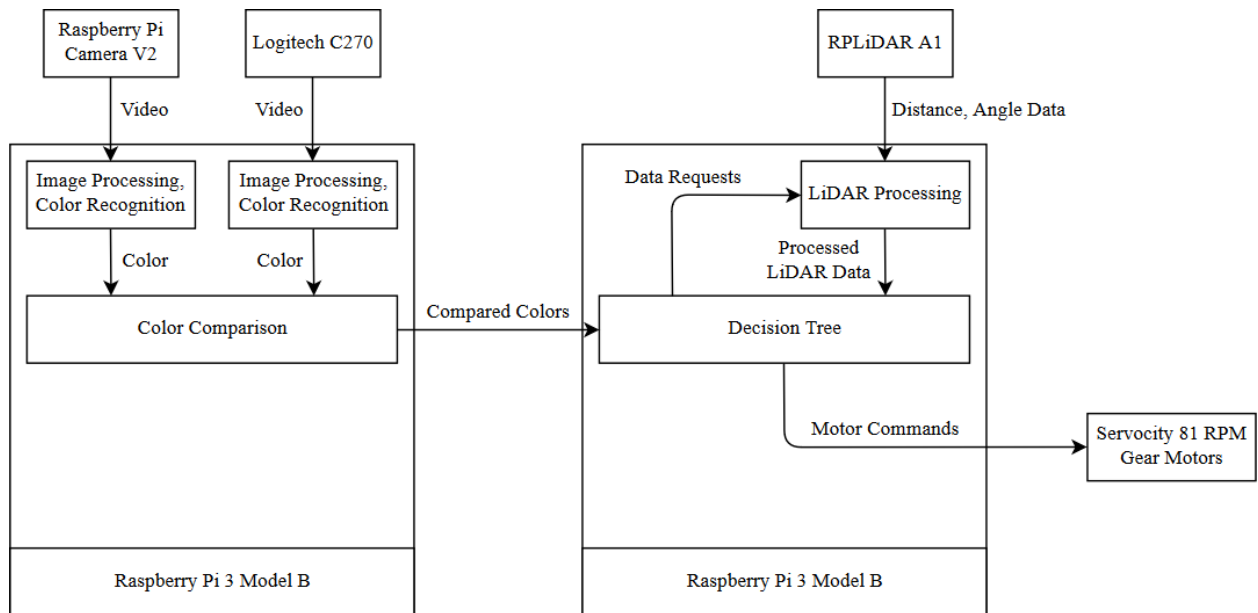


Figure 5. Block Diagram of Robot.

this is done, the connection of hardware and flow of data through the robot should be observed.

This may be seen in the block diagram shown in Figure 5.

Of the elements present in the block diagram, the only element not previously discussed at length is the decision tree block. This portion of the robot consists of the software control portion of the robot that was previously mentioned. Development of this portion of the robot began with the creation of code that would allow the robot to exit the corner that it would be placed in at the start of a round. This code was relatively simple and functioned by directing the robot to turn until the LiDAR's 0° distance measurement was reported as being less than 1800 mm. That length was approximately the distance between a corner of the course and the closest point on the central structure's wall. Once the robot had stopped turning, it would provide equal power to the motors and move towards the central structure in a straight line until the LiDAR reported that the 0° distance measurement was less than a given distance. The exact value used for the distance varied throughout testing, and ultimately would not matter, as this method of leaving the corner

was replaced over Spring Break. The reason for its replacement was because if the robot turned off course for any reason, it was possible for the  $0^\circ$  measurement from the LiDAR to miss the central structure and hit one of the course walls instead. This meant that the robot would stop only after it had neared the wall. While attempts to fix the function had been made by having a swathe of front-facing LiDAR readings be sampled, the weight balance and motor troubles still could cause the robot to veer notably off course. This issue was further exacerbated if there were any objects between the robot and the central structure, as their friction would drag the robot off course if they did not stay in the center of the collection area.

The function that replaced the original “move to center” function worked in a similar fashion. However, instead of referring to LiDAR readings in front of the robot, the function referred to a  $10^\circ$  wide slice of LiDAR readings centered on the  $180^\circ$  angle, which was directed behind the robot. As such, the robot would move forward until the minimum distance reading of the aforementioned slice of angles was 1050 mm from the corner that it had started in. This prevented the imperfect movement of the robot from having a major impact on its ability to leave the corner, as its movement away from the corner would result in travelling more than 1050 mm. While this distance may seem small compared to the 1800 mm separating a corner from the central structure, the distance was chosen partially because experience with the robot revealed that delays in code processing would result in it not having instantaneous response to LiDAR readings. Thus, the robot would travel some distance greater than 1050 mm before stopping. An additional change that was implemented in this “move from corner” code was that it no longer called for the robot to turn towards the center. Instead, the robot was placed facing the center, as this was discovered to be permissible within the robotics competition.

After the robot had moved out of its corner, it was instructed to conduct a right zero-point turn until the left side of the robot was facing the center. This was determined to have happened if the minimum distance read by the LiDAR in a  $30^\circ$  wide slice centered on the  $290^\circ$  angle was greater than 800 mm. This allowed the robot to turn until it was roughly parallel with the central structure, after which it would stay still for 2 seconds. After these two seconds, it would record the color present on the side of the central structure closest to the robot. The robot was kept still during this time to ensure that the image processing code had time to receive clear video from the robot's left-facing camera. This was done to enable the robot to return to the corner that it had started in, as it would do so by comparing the color of the quadrant that it was in to its starting color and doing so would have enabled the robot to earn extra points under certain conditions.

After the robot had recorded its starting region color, it would begin to travel around the central structure in an orbit. Three distinct orbits were defined, with each orbit having a unique distance to maintain from the center and unique motor speeds. The distances were initially defined as 140 mm, 420 mm, and 700 mm, and were based on the idea of having the robot orbit the course at three distinct distances that would allow for all objects to be collected after several orbits. The motor speeds were selected to allow the robot to travel in an orbit around the center with minimal need to correct its orbit. However, implementation of these parameters and experimentation soon demonstrated that such an implementation would be difficult to achieve. For example, given delays in the processing of code on the Raspberry Pi 3, the robot would often begin a close orbit of the center, only to begin to exit the orbit. The robot would attempt to correct this by conducting a zero-point turn, but once again, delays in code processing would cause the robot to not stop its turn quickly enough, thus causing it to turn towards the center and

eventually collide with it. In addition to that, the robot could not respond quickly to getting too close to the central structure and attempts to fix this issue did not resolve it. As such, it was decided that the innermost orbit would not purposefully be used. This decision was reinforced when it was also observed that the robot would weave in and out of its orbits, meaning that it was able to collect objects that lay outside a particular orbit.

With this in mind, and after experiencing further difficulties in having the robot maintain distinct orbits, it was decided that the robot would maintain an orbit between 300 mm and 345 mm of distance from the central structure. With careful tweaking of what groups of LiDAR readings would be referred to, the robot was able to maintain an uneven orbit that would take it close and far from the central structure while usually avoiding collision with the central structure and the space hotels arranged around it. To ensure that such collisions would not occur, the robot was instructed to adjust its motor speeds such that it would move more strongly towards the right if it got within 300 mm of the central structure and more strongly to the left if it went more than 345 mm from the central structure.

The robot's ability to avoid collision was further strengthened with the inclusion of code that would instruct it to move its collected objects to a corner if it detected that the color of the quadrant that it was in matched the majority color of the objects that it collected. As noted earlier, while the robot's color recognition was not perfect, it achieved a satisfactory level of success. This movement to the corner was accomplished by having the robot constantly compare the colors of its collected objects and the color detected by its central-structure-facing camera. If these colors matched, the robot would make a right turn. The turn would stop after the robot detected that any distance in an  $8^\circ$  wide slice centered on  $14^\circ$  was less than any distance in an  $8^\circ$

wide slice centered on  $24^\circ$ . This comparison took advantage of the fact that the outer walls of the course formed a square, and the robot would only find the aforementioned comparison true when it began to turn past a corner.

After the robot had stopped turning and was roughly facing the corner, it would move forwards, and would adjust its motor speeds to keep it centered on the corner based on a comparison of distance measurements. The distances measured were those of  $10^\circ$  and  $350^\circ$ . If the  $10^\circ$  measurement was larger, then the robot would be facing to the left of the corner and would need to move right. Likewise, if the  $350^\circ$  measurement was larger, then the robot would be facing to the right of the corner and would need to move left. This comparison and movement to the left and right to remain centered continued until the robot was within a 300 mm of the corner, after which it would stop and reverse back into orbit. Stopping 300 mm from the corner would ensure that the objects collected by the robot were left in the corner, thereby maximizing the points scored. The robot would reverse one of 3 set distances away from the corner. These distances were 1000 mm, 900 mm, and 800 mm, and were chosen because experimenting showed that they would positively alter the orbit of the robot. The robot would cycle through these distances, with the first reversal out of a corner using 1000 mm, the second reversal 900 mm, and the third reversal 800 mm. After the third reversal, the robot would start through the set again. The distances were compared to the  $0^\circ$  LiDAR distance measurement.

The robot would continue orbiting the central structure, collecting objects, and depositing objects when it determined that such was appropriate for 150 seconds. After 150 seconds, it would continue its orbit and collection of objects, but would now only go to a corner when it found itself in the quadrant that it had started in. This was determined by comparing the color















detected by the robot's left-facing camera to the color detected at the start of its movement. If the colors matched, the robot would engage in the aforementioned corner movement code, although it would no longer reverse out of the corner. Instead, when it reached a distance of 300 mm from the corner, it would stop. This behavior was included with the intent to have the robot raise a flag in its starting square, which would have allowed it to earn 25 points, as noted in Appendix A. However, as it turned out, space and time constraints would prevent the flag-raising system from being implemented. The code associated with the control schema described may be found in Appendix E.

## **VII. CONCLUDING ANALYSIS**

Over the course of development and testing, the most consistent performance that was achieved was the scoring of approximately 160 points on average. This was usually the result of the robot earning 5 points for leaving its starting square, 5 points for entering the central area, 15 to 20 points for 3 or 4 complete counterclockwise orbits of the central structure, and 130 points for the removal of objects from the central area to the corners. The number of objects that were correctly placed based on the color varied widely, as did the number of objects removed. Sometimes the robot managed to remove 8 or 9 objects from the central area, although it would usually only manage 6 or 7 objects. Sometimes the arrangement of objects would allow the robot to easily color-match most objects with their colored corners, although more typical behavior was the color matching of 2 to 4 objects. At the competition, the robot earned 10 points in the first qualifying round and 135 points in the second qualifying round, making it the 12<sup>th</sup> most successful robot at the competition out of 46 robots. However, this was not enough to have the

robot compete in the 3 tournament style rounds at the end of the competition. For the robot to have been able to do so, it would have needed to earn at least 176 points, as demonstrated in

Rank		University	Qualifying Round 1	Qualifying Round 2	Sub-Total
1		Lipscomb University	330	335	665
2		North Carolina State University	375	30	405
3		University Of Tennessee - Chattanooga	235	145	380
4		University of Alabama - Tuscaloosa	155	170	325
5		Clemson University	130	110	240
6		Mississippi State University	75	155	230
7		University of West Florida	35	180	215
8		Virginia Military Institute	35	175	210
9		University Of Memphis	115	90	205
10		Bethune-Cookman University	100	60	160
11		Tennessee Technological University	10	140	150
12		Murray State University	10	135	145

*Figure 6. Rankings of 12 best-performing robots at the robotics competition.*

Figure 6. Several figures of the scores earned at the IEEE robotics competition by every team may be found in Appendix C, from Figure 12 to Figure 15.

Ultimately, the robot demonstrated rather variant performance. The notably below-average performance of the robot in the first qualifying round was likely a result of the fact that at the time the robot was running code that had been worked on overnight. This code was not truly proven in its ability to perform, and it demonstrated that by earning only 10 points. The below-average performance in the robot in the second qualifying round was likely due to variance in the course setup and the specific operation of the robot in that round compared to its average operation during testing. An explanation for this would be that the Raspberry Pi 3 Model B microcontroller not having enough processing power to run the controlling code efficiently,

meaning that the robot's ability to respond to sensor input was hampered. Additionally, the test course that had been using differed from the courses that were used in the official competition on account of having been made with speed in mind, although there had also been issues with procuring the right materials for the test course because of them not being present in Murray. If the test course had been more properly constructed, the results of testing would have likely been more analogous with the results from the competition.

With this in mind, there are likely many positive changes could have been made to the robot and the robot development process. As noted before, one such change would have been the use of hardware that had greater processing power. This likely would have taken the form of using a small form factor computer in the robot, rather than a Raspberry Pi microcontroller. Such a computer could have been equipped with a processor with more cores and greater processor frequency, which would have enabled the more efficient execution of code and faster response to pertinent sensor input. Another change would have been returning to an open structure for the robot, as was originally designed. This would have allowed for much easier maintenance and modification to be done to the robot, as there would have been more room to move around components and the wires connecting them. A third positive change would have been searching for motors that provided both more torque and greater rotational velocity, even if they had cost more. This would have allowed the robot to maneuver about the course with greater speed, while suffering from less loss of speed whenever it had to push objects around the course. Furthermore, a higher torque would have prevented the robot from experiencing motor stalling due to collecting too many objects. Another improvement would have been the usage of wheels with a greater ground contact area and a surface with more gripping ability, as this would have allowed

the robot to achieve more traction on the course and might have mitigated any instances of wheels slipping.

Having used a LiDAR with better specifications, such as the RPLiDAR A2 or A3, would have benefitted the robot. While such LiDARs would have cost more, they likely would have also mitigated a notable issue with the RPLiDAR A1, which was that it never produced a full array of 360 distance measurements. Instead, it would usually create 250 to 300 distance measurements, which caused slight difficulty in using it for course detection and position tracking. This lack of data was likely the result of either some form of built-in error-correction, wherein unsatisfactory measurements would be removed, or some inability for all of the data points to be collected by the code that took in the LiDAR data on the Raspberry Pi. A final improvement to the robot itself would have been to make use of a PID controller that operated on the basis of maintaining a set distance from the closest point on the central structure. While this would have been more complex to implement than the manipulation of geometry that was done, if correctly done it would have allowed for many of the original ideas regarding the robot to be implemented. Such a development would have also likely required the usage of encoded motors that could determine how far travel, which would have increased the motor price.

With regards to the actual practice of robot development, the most notable potential improvement would have been that of making greater use of human resources present at Murray State University. For example, as noted, implementing a PID controller would have likely been somewhat challenging. However, Dr. Bunget of the Institute of Engineering has previously taught Control Systems classes, wherein the topic of PIDs have been discussed. Furthermore, Dr. Bunget is well versed in the usage of Python. If human resources had been used more efficiently,

then Dr. Bunget may have provided guidance and instruction regarding the implementation of a PID in the style that was desired. This would have improved the performance of the robot.

Another notable improvement to the practice of robot development would have been conducting rigorous testing during the competition, as this would have revealed that the code that was used for the first qualifying round was prone to failure. Furthermore, focus should have been placed on only changing one element of the code at a time. This would have permitted the observations of only the effects caused by changing that single element. Lastly, it should have been realized that trying to code early in the morning after having had little sleep ultimately leads to poor results, such as the code that resulted in the robot's failure in the first qualifying round.

## VIII. REFERENCES

- [1] IEEE, “Hardware Competition Rules,” *IEEE*, 2019. [Online] Available: [http://sites.ieee.org/southeastcon2019/files/2019/02/2019-SoutheastCon-HW-Rules\\_v1\\_3-1.pdf](http://sites.ieee.org/southeastcon2019/files/2019/02/2019-SoutheastCon-HW-Rules_v1_3-1.pdf) [Accessed 7 Apr. 2019].
- [2] S. Modi, P. Chandak, V.S. Murty, and E.L. Hall, “A Comparison of Three Obstacle Avoidance Methods for a Mobile Robot,” *University of Cincinnati*, 2001. [Online]. Available: <https://ceas.uc.edu/content/dam/ceas/documents/UC%20Center%20for%20Robotics%20Research/Sachin2001c.doc>. [Accessed: Nov. 19, 2018].
- [3] E. Olson, “A Primer on Odometry and Motor Control,” *MIT OpenCourseWare*, 2004. [Online]. Available: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-186-mobile-autonomous-systems-laboratory-january-iap-2005/study-materials/odomtutorial.pdf>. [Accessed: Nov 19, 2018].
- [4] A. Krizhevsky, I. Sutskever, and G.E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Proc. of the 2012 Neural Information Processing Systems Conference, NIPS 2012, 3-8 Dec. 2012, Stateline, Nevada* [Online]. Available: NIPS Proceedings<sup>β</sup>, <http://papers.nips.cc/>. [Accessed: Nov. 19, 2018].
- [5] PowerStream, “How to Calculate Battery Run-Time,” *Powerstream*. [Online]. Available: <https://www.powerstream.com/battery-capacity-calculations.htm>. [Accessed: Nov. 19, 2018].

## **APPENDIX A. RULES, SPECIFICATIONS, AND POINTS**

### **I. Playing Field**

1. The playing field (see ATTACHMENT A) will be based on a square carpet base with 2.74-m (108") sides surmounted by walls with internal dimension of 244-cm x 244-cm (96 1/8" x 96 1/8") that extend 30-cm (11 7/8") above the playing field surface. The walls are no longer painted, the wall material comes in a pre-primed flat white finish.
2. The playing field will be divided into two zones, Zone 1: an area outside of the orbital line where robot corner squares are located and Zone 2: an area within the orbital line surrounding a central column.
3. The playing field will be further subdivided into four equal quadrants. In order to assist robots in identifying the quadrants, they will be separated with two color-coded lines (see ATTACHMENT A).
4. The division between quadrants consists of a pair of adjacent, color-coded, lines separating the quadrants which includes one WHITE line paired with one each of RED, YELLOW, BLUE, and GREEN (counter-clockwise order). The color of the separating line corresponds to each corner square.
5. The playing field will include four 30-cm square robot corner squares located in the playing field corners. The corner squares will be separated from Zone 1 by a 4.78-cm (1.88") wide WHITE line. Each corner will be marked by a color-coded corner post in RED, YELLOW, BLUE, and GREEN.
6. Zone 1 will be separated from Zone 2 by a 2.54-cm (1") wide WHITE orbital line as depicted in ATTACHMENT A.
7. The playing field will also include four flashing LED lights that will represent geostationary Spacetels. These items will be placed on the orbital line separating Zone 1 and Zone 2.
8. The playing field environment lighting is not specified nor controlled. The intended location of the hardware competition will be in a high bay/ceiling convention center
9. No restrictions will be placed on spectators and those present during the competition for use of their cameras and cellphones. As a result, flash photography, infrared range finders on cameras and camcorders will be permitted. Intentional interference with the operation of the robots is not allowed and will result in sanctions.

### **II. Playing Field Objects**

1. The playing field will contain a total of twelve (12) space debris objects. Eight color-coded 5.08-cm (2") wooden cubes and four 6.35-cm (2 1/2") diameter color-coded pit balls representing space debris. The placement pattern will be RANDOM but limited to only within Zone 2. Spacetels will be represented by flashing lights (ATTACHMENT A).
2. Eight of the debris objects will be 5.08-cm (2") wooden cubes two of each color RED, YELLOW, BLUE, and GREEN.
3. Four of the debris objects will be 6.35-cm (2 1/2") diameter pit balls one of each color RED, YELLOW, BLUE, and GREEN.
4. There will be four satellite objects representing Spacetels. These will be placed on the white orbital line circle intersecting the quadrant lines, in a fixed position. The Spacetels will have flashing amber LEDs.

### III. Robot Specifications

1. The robot must operate completely autonomously once started.
2. The robot must be entirely self-contained, including any power source.
3. The maximum size of the robot will be restricted to 22.86-cm (9") x 22.86-cm (9") x 27.94-cm (11"). These dimensions are in order of L x W x H.
4. This maximum size applies when the robot is in the starting square at the start of a round or is in motion on any part of the playing field during the round.
5. When not in motion, the robot may extend a maximum of 7.62-cm (3") by 7.62-cm (3") in length and width direction at a time.
6. The extension must be physically connected to the main robot at all times.
7. The robot may not exceed beyond its maximum allowable dimension and may not include any detachable or remoted extensions.
8. There is no weight limit nor construction material restriction with exception to safety and security rules.
9. Any part of the robot that is deemed by contest officials to be dangerous or injurious to the participants, audience, staff, playing field or surroundings will result in disqualification. If in doubt, ask in advance.
10. Pyrotechnics, compressed gas, hydrocarbons, toxic or corrosive materials are not allowed.
11. The on-board flag must contain the school logo (if affiliated with a university), State, territory or US flag.
12. A robot may not operate in a manner that excessively damages the playing field, causing stoppage of the competition, or require repair of the field for the next competition.
13. Each robot must have a bumper that surrounds 80% of its perimeter in a continuous stretch. This bumper must be the outermost structure at all times when the robot is moving.
14. The bumper must present a vertical surface at least 2.54-cm (1") high and cover, at a minimum, the space from 3.81-cm (1 ½") to 6.35-cm (2 ½") above the playing field.
15. The bumper may be of any shape around the robot and need not be outwardly convex on all surfaces but must not have any radius of curvature less than 1-cm.
16. The bumper must be included in the maximum 22.86-cm (9") x 22.86-cm (9") x 27.94-cm (11") (L x W x H) overall size.
17. In addition to meeting safe operation requirements, a robot will need to pass the size qualification.
18. Robots may be modified physically, reprogrammed, and/or recharged between each match. However, any physical modification will require a re-inspection for safety and overall size compliance.
19. Robots that do not meet these requirements will not qualify and will not be allowed to compete.

### IV. Rules

1. Robots must be completely autonomous.
2. Robots must be self-contained and remain a single unit (cannot break apart).



3. Robots may be no larger than 22.86-cm (9”) x 22.86-cm (9”) x 27.94-cm (11”) (L x W x H) at the start or while in motion.
4. A size and safety qualification inspection will be done prior to each round of play.
5. Robots must not damage the playing field, halt competition or require repair of the playing field. Doing so will result in either point deduction or disqualification depending on the severity.
6. There are two qualifying rounds followed by a quarter-final, semi-final, and final round.
7. The scoring for each qualifying round is cumulative.
8. The top 8 scoring teams from the qualifiers will move on to the quarter-final round.
9. The quarter-final round consists of 4 matches.
10. The semi-final round consists of 2 matches.
11. The final round consists of a single match.
12. During the two qualifying rounds, there will be only one robot on the playing field.
13. During the quarter-final, semi-final, and final rounds (see B), there will be two robots competing on the playing field at the same time.
14. Prior to the beginning of each round, the next team(s) to play will be announced. They must present their qualifying robots and place them on their designated corner square (home base) within one minute of their announcement. Missing the announcement deadline will result in disqualification of the missing team(s) for that round.
15. For each round, home bases will be randomly selected. For final rounds, the opponent will be placed on the opposite corner from the first randomly selected home base.
16. A hands-off period will follow the placement of a robot on the playing field. During this time, twelve (12) objects will RANDOMLY be placed in Zone 2 on the playing field.
17. After the objects have been placed on the playing field, a contest official will give a verbal start command. A team member will then manually start the robot by pressing a button or flipping a switch on the top of the robot. No further interaction between the robot and a team member is allowed during a round.
18. Team members are not allowed to enter the playing field or touch their robot until the three-minute match ends or a team decides to terminate its participation.
19. Any points scored until early play termination for a team will count towards the final point tally for that round.
20. A round is three minutes from the point that the start command is given until a buzzer sounds the end of a round.
21. Upon start of the match, the robots will depart from the home base and enter Zone 1.
22. Manipulation of the objects are left for each team to decide. The goal of the match will be to de-orbit as many objects from Zone 2 to Zone 1 (which includes corner squares) as possible.
23. If a robot carrying an object is in any part of a corner square at the end of play, all the objects it carries will score.
24. During final rounds, debris objects placed in an opponent’s assigned squares will score for the opponent.
25. The contest’s judge decision is final regarding whether a block is in scoring position or not.

26. Destructive Interference: A team may not take any action that purposely interferes with the course of play or causes damage to the playing field or competing robot. Based on Judges' opinion, if the robot is damaging the field of play or interferes with operation of opposing robot, the impeding robot will be subject to disqualification for that match.
  27. Teams should use caution in filing appeals. Unsuccessful appeals will cause a 40 point deduction from the Team filing the appeal if, after review, the Judges determine that the original scoring was correct.
  28. Judges decisions are final.
  29. Violations of IEEE code of ethics and code of conduct will not be tolerated and will result in point deduction, disqualification, or ejection from the event based on the severity of the violation.
  30. When addressing judges with questions teams are expected to act within the IEEE code of conduct. Only one designated team captain can address the judges for written or verbal decision appeals.
- V. Judging and Scoring
1. Points may be earned by:
    - a. Exiting the starting base and entering Zone 1
    - b. Crossing over from Zone 1 and into Zone 2
    - c. Orbiting the playing field in a counter-clockwise direction in Zone 2
    - d. Removing orbital debris from Zone 2 to Zone 1
    - e. Delivering orbital debris to a corner square
    - f. Color-matching debris to a corner square
    - g. Finishing in the assigned home base
    - h. Raising an on-board flag
  2. Points awarded per task are as follows:
    - a. Five (5) points are awarded to a robot that completely leaves home base.
    - b. Five (5) points are awarded for the first time a robot enters Zone 2 (the orbital circle).
    - c. Five (5) points are awarded for every complete, counter-clockwise orbit of the playing field.
    - d. Ten (10) points are awarded for each debris removed from Zone 2, and an additional ten (10) points for blocks placed in corner squares (total of 240 points).
    - e. If a robot places a color matched debris in the associated color corner square, ten (10) additional bonus points will be awarded (total of 120 bonus points)
    - f. Ten (10) points are awarded for a robot residing in its home base at the end of a round.
    - g. Twenty-Five (25) points are awarded to a robot that raises an on-board teams' school, State, territory or US flag at the end of a round and while in home base.
    - h. If a robot carrying an object is in any part of the home square at the end of play, all the objects it carries will score.
    - i. A penalty of ten (10) points will be subtracted from a positive score for every separate collision of a Spacetel object.

3. During the playoff rounds
  - a. Each playoff team will be given assigned squares, these are the home base and the first counter clockwise corner square, as scoring squares.
  - b. Points are awarded as in the qualifying rounds, both for debris in corner squares and with a bonus for color matching.
  - c. Color matching bonus points apply to both corner squares of the assigned squares during playoff rounds.
  - d. Points are also awarded for moving a debris object from Zone 2 to Zone 1 in both quadrants of the assigned squares, the home base quadrant and the first counter clockwise corner square quadrant.
  - e. If a debris objects placed in an opponent's assigned squares will score for the opponent
4. The contest's judge decision is final regarding whether a block is in scoring position or not.

Attachment A

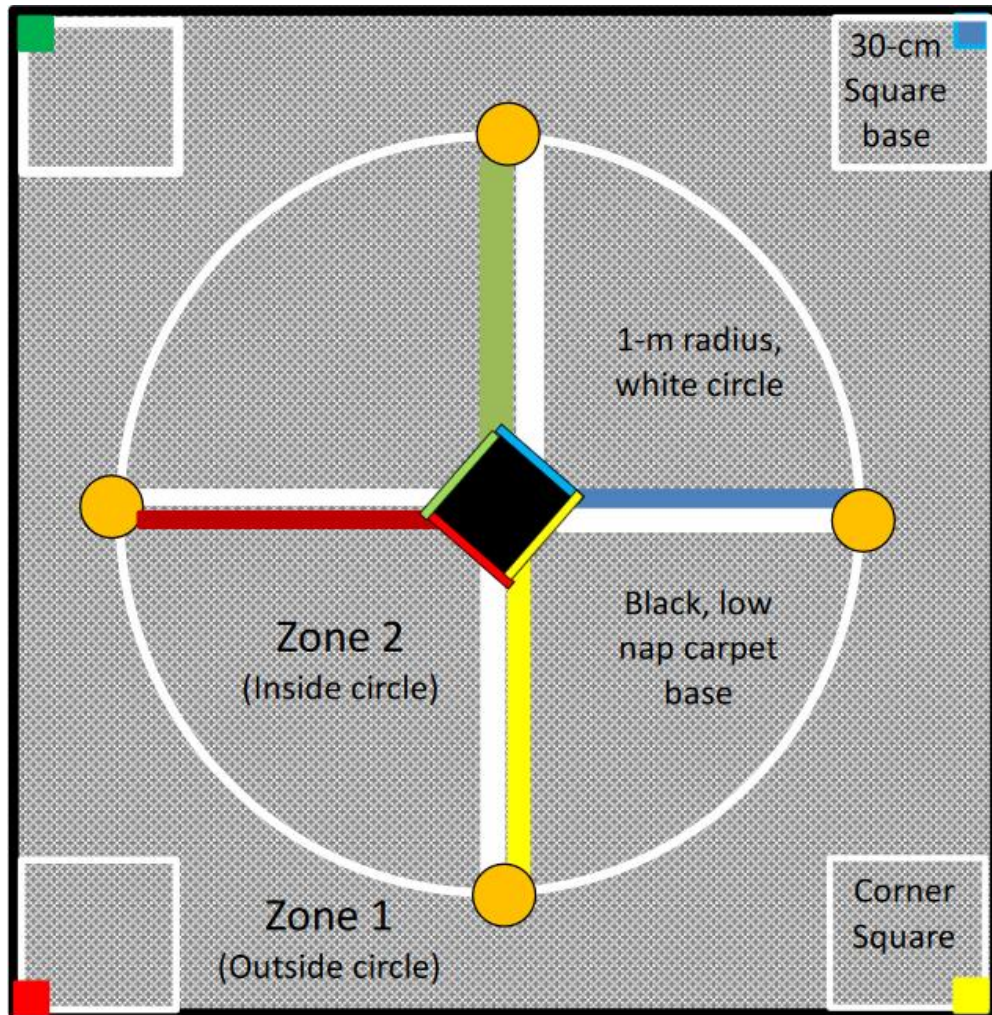


Figure 7. Attachment A, the course setup diagram.

Attachment B

1 <sup>st</sup> Rank Team			2 <sup>nd</sup> Rank Team
	Winner A	Winner B	
8 <sup>th</sup> Rank Team			7 <sup>th</sup> Rank Team
3 <sup>rd</sup> Rank Team			4 <sup>th</sup> Rank Team
	Winner C	Winner D	
5 <sup>th</sup> Rank Team			6 <sup>th</sup> Rank Team

**Semi-Final Round Ladder**

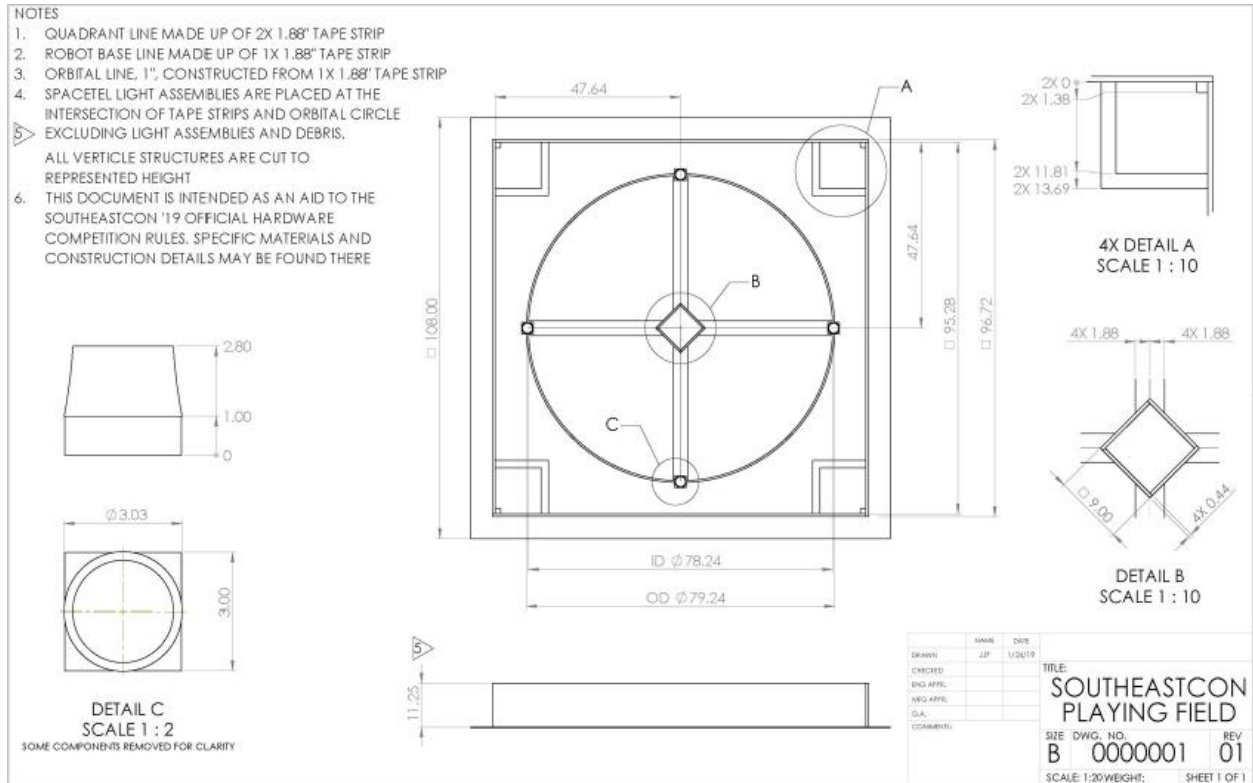
Winner A			Winner B
	<b><u>Winner A'</u></b>	<b><u>Winner B'</u></b>	
Winner D			Winner C

**Final Round Ladder**

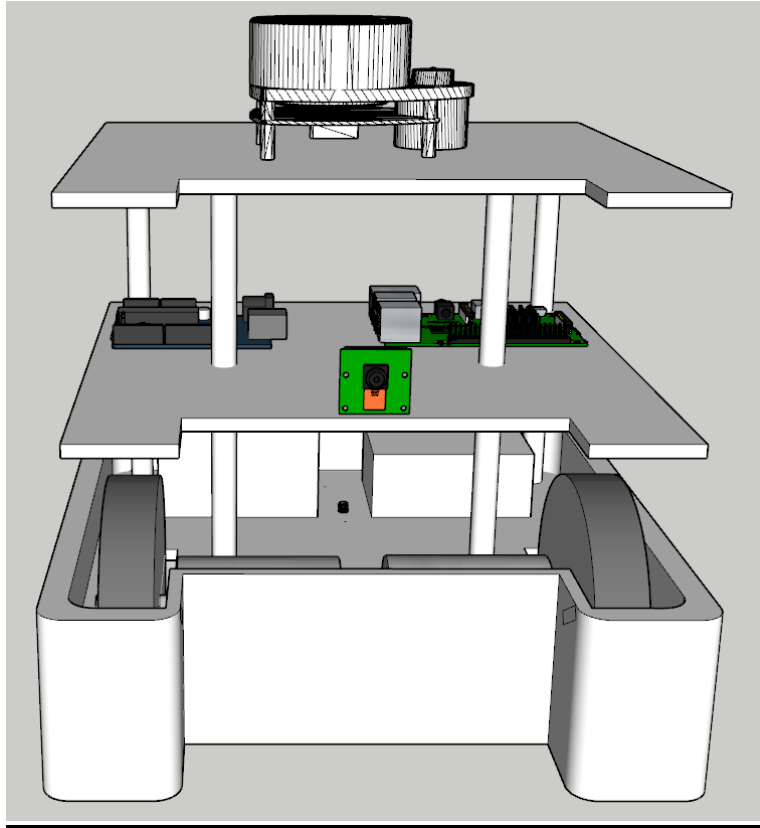
Winner A'	<b>First Place Winner</b>	Winner B'
Loser A'	<b>Third Place Winner</b>	Loser B'

*Figure 8. Attachment B, the competition ladder.*

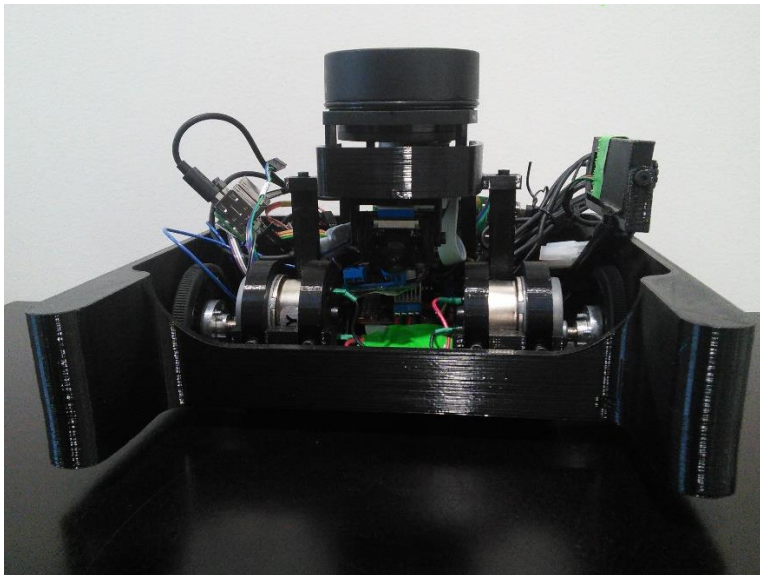
## APPENDIX B. ADDITIONAL FIGURES AND TABLES



*Figure 9. Blueprint of the course.*
















*Figure 10. Additional front view of robot.*



*Figure 11. Front view of final robot.*

## APPENDIX C. FINAL COMPETITION SCORES

Rank ▼		University	Check-In	Inspection	Qualifying Round 1	Qualifying Round 2	Sub-Total
1		Lipscomb University	✓	✓	330	335	665
2		North Carolina State University	✓	✓	375	30	405
3		University Of Tennessee - Chattanooga	✓	✓	235	145	380
4		University of Alabama - Tuscaloosa	✓	✓	155	170	325
5		Clemson University	✓	✓	130	110	240
6		Mississippi State University	✓	✓	75	155	230
7		University of West Florida	✓	✓	35	180	215
8		Virginia Military Institute	✓	✓	35	175	210
9		University Of Memphis	✓	✓	115	90	205
10		Bethune-Cookman University	✓	✓	100	60	160
11		Tennessee Technological University	✓	✓	10	140	150
12		Murray State University	✓	✓	10	135	145
13		University of the West Indies - Mona	✓	✓	10	95	105

*Figure 12. Scores of 1<sup>st</sup> to 13<sup>th</sup> highest scoring teams.*
















Rank		University	Check-In	Inspection	Qualifying Round 1	Qualifying Round 2	Sub-Total
14		University of Southern Mississippi	✓	✓	40	45	85
14		University of Tennessee - Knoxville	✓	✓	0	85	85
16		Auburn University	✓	✓	40	40	80
16		University of Kentucky	✓	✓	45	35	80
16		Virginia Tech	✓	✓	75	5	80
19		Mercer University	✓	✓	20	50	70
20		Florida Institute of Technology	✓	✓	10	55	65
21		University of Central Florida	✓	✓	30	25	55
21		University of Southern Indiana	✓	✓	30	25	55
21		Valencia College	✓	✓	55	0	55
24		Georgia Southern University	✓	✓	40	5	45
25		Pellissippi State Tech Com College	✓	✓	10	30	40
25		University of North Carolina - Charlotte	✓	✓	0	40	40
27		Eastern Carolina University	✓	✓	30	5	35

Figure 13. Scores of 14<sup>th</sup> to 27<sup>th</sup> highest scoring teams.

















Rank ▼		University	Check-In	Inspection	Qualifying Round 1	Qualifying Round 2	Sub-Total
28		Bob Jones University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	20	10	30
29		University of Florida	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	15	10	25
29		University of South Florida	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5	20	25
31		Florida A&M/Florida State University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	10	20
31		Western Kentucky University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	10	20
33		Florida Atlantic University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5	10	15
33		Western Carolina University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	5	15
35		Christian Brothers University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	10	10
35		Florida A&M University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	0	10
35		Florida International University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	10	10
35		The Citadel School of Engineering	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	10	10
35		Virginia Commonwealth University	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	0	10
40		AAMU	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0
40		Georgia Institute of Technology	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5	0	5

Figure 14. Scores of 28<sup>th</sup> to 40<sup>th</sup> highest scoring teams.




Rank ▼		University	Check-In	Inspection	Qualifying Round 1	Qualifying Round 2	Sub-Total
42		Embry-Riddle Aeronautical University	<input type="checkbox"/>	<input type="checkbox"/>	0	0	0
42		UAH	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0
42		University of North Carolina - Asheville	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0

Figure 15. Scores of 42<sup>nd</sup> highest scoring teams.

## APPENDIX D. CALCULATIONS

### Calculation of Minimum Ampere-Hours of Robot Battery

Maximum Current Draws of Robot Electronics			
Device	Maximum Current Draw (A)	Quantity	Total Maximum Current Draw (A)
Raspberry Pi 3	2.5	1	2.5
RPLiDAR A1	0.35	1	0.35
Arduino	0.10	1	0.10
Wheel Motor	0.15	2	0.30
<b>Total</b>			3.25

$$C = I * T = 3.25 \text{ A} * 1 \text{ hr} = 3.25 \text{ Ah}$$

$$C' = \frac{C}{1.0} = \frac{3.25 \text{ Ah}}{1.0} = 3.25 \text{ Ah}$$

$$C'' = \frac{C}{0.5} = \frac{3.25 \text{ Ah}}{0.5} = 6.5 \text{ Ah}$$

## **APPENDIX E. CODE**

```
import scan
import Motor
import pigpio
import numpy as np
import time
import os
import LidarPrint as lp
import serial
import math
from time import sleep
import threading
from LidarPrint import *
import socket
import os
import subprocess
from subprocess import Popen, PIPE
import shlex
import ctypes

ser_color = serial.Serial("/dev/serial0",115200)

##### Motion #####
# Initiates the motor pins, then sets the motor profiles and inverts the left
# motor, as this was needed for both motors to go the same direction. It
# should be noted that the left motor referenced in the code was the right
# motor and vice versa.

#Motor A
mlin1= 26
mlin2= 19
m1PWM = 13
#Motor A
m2in1 = 21
m2in2 = 20
m2PWM = 16

left = Motor.Motor(m1PWM,mlin1,mlin2)
right = Motor.Motor(m2PWM,m2in1,m2in2)
left.Inverse()

##### LiDAR #####
# Initiates the LiDAR connection to the Raspberry Pi 3 microcontroller and
# sets the methods for referencing the data transmitted over the connection

port = "/dev/ttyUSB0"
ser_lidar = serial.Serial(port, 115200, timeout = 5)

ser_lidar.setDTR(False)
print (ser_lidar.name)
lidar = Lidar(ser_lidar)
```

```

'''sensors = ser_color.readline().rstrip()'''
'''data = sensors.split('^')'''

##### Methods #####

def data(lidar_data):
    x = lidar_data
    x[np.where(x[:,0] == 0),0] = 3000
    return x

# Takes in a given set of LiDAR data and takes any empty readings (equal to
# 0) and sets them to 3000, to avoid errors in movement logic.

def distance_slice(lidar_data, angle, spread):
    x = lidar_data[:,1] - angle
    x = np.abs(x)
    minim = np.where(x == np.min(x))[0][0]
    low = minim-spread
    high = minim + spread
    dist = lidar_data[low:high,0]

    if dist.size:
        abc = 0
    else:
        dist = 3000 * np.ones((spread*2))

    dist = np.asarray(dist)

    return dist

# Takes a given set of LiDAR data, an angle, and a set spread, and returns a
# data set centered on the provided angle. "Spread" determines the number of
# array indices on each side of the angle. This method was created to counter
# the fact that the LiDAR would not provide a full 360 degrees of readings,
# meaning that one could not directly reference indices by inputting the
# angles that they would have corresponded to.

def stop():
    right.Stop()
    left.Stop()

# Stops the motors.

def read():
    x = data(lidar.getPoints(ser_lidar, polar = True))
    while (x.size < 100):
        x = data(lidar.getPoints(ser_lidar, polar = True))
    return x

# Reads in LiDAR data

def straight():
    right.Drive(230,0)
    left.Drive(231,1)

# Sets motor speeds for the robot to travel directly forward.

```

```

def reverse():
    right.Drive(230,1)
    left.Drive(230,0)

# Sets motor speeds for the robot to travel directly backward.

def rightZP():
    right.Drive(150,0)
    left.Drive(150,0)

# Sets motor speeds for the robot to perform a zero-point right turn.

def leftZP():
    right.Drive(150,1)
    left.Drive(150,1)

# Sets motor speeds for the robot to perform a zero-point left turn.

def colorMethod():
    msg = ser_color.read(3)
    front = int(msg[0])
    side = int(msg[2])
    final = np.array([front, side])
    return final

# Reads in color data from the two cameras on the robot and outputs an array
# of the color values.

def colorCorner(colorData):
    if colorData[0] == colorData[1]:
        return 1
    else:
        return 0

# Compares an input array of color data (usually provided by 'colorMethod')
# and returns 1 or 0 depending on whether both values of the array match.

##### Control Logic #####
# Determines the actions taken by the robot depending on a number of
# conditions evaluated by the robot's sensors.

# Cases
# Case 0: the robot moves out of the corner and turns to be parallel
# with the central structure.
# Case 1: the robot orbits around the central structure.
# Case 2: the robot moves to a corner to deposit objects, after which it
# returns to case 0. If enough time has passed, it enters case 3.
# Case 3: the robot stops and raises a flag (ultimately unrealized)

start = time.time()
orbitTime = 150
end = 0

# Sets the variables for tracking and comparing robot runtime, with a maximum
# runtime of 150 seconds set.
orbit_group = 0
corner_count = 1

```

```

case = 0

# Initiates the variables for tracking which orbit the robot is in and how
# many times it has travelled to a corner. Also sets the robot to case 0.

while True:
    end = time.time()

    if case == 0:
        begin_dist = np.array([1050, 900, 800])

# Initiates the array containing distances for robot to move from wall

    straight()

# Instructs the robot to go forward

    x = read()
    angle = 180
    width = 5
    dist = distance_slice(x, angle, width)
    while np.min(dist) < begin_dist[orbit_group]:
        x = read()
        dist = distance_slice(x, angle, width)
    rightZP()

# The robot continues forward until It has travelled 1050 mm from the wall
# behind it, after which it does a right zero-point turn.

    x = read()
    angle = 250
    dist = distance_slice(x, angle, 15)
    while(np.min(dist) < 800):
        x = read()
        dist = distance_slice(x, angle, 5)
    while(np.min(dist) > 800):
        x = read()
        dist = distance_slice(x, angle, 5)
    stop()
    sleep(2)

# The robot turns right until it is roughly parallel with the closest edge of
# the central structure, after which it stops and sleeps for 2 seconds.

    col = colorMethod()
    while col[1] == 4:
        col = colorMethod()
    start_col = col[1]

# The robot reads in colors from cameras, and sets the color reported from
# the side camera as the 'start_col' (starting central color). The robot
# calls 'colorMethod' again if it reads a '4' due to '4' being associated
# with non-applicable colors (colors that are not red, green, blue, or
# yellow.

    right.Drive(210,0)
    left.Drive(215,1)

```

```

        case = 1

# Instructs the robot to travel forward while slowly turning to the left, so
# it may orbit the central structure.

##### Case 1 #####

    if case == 1:

        slice_width = np.array([15, 20, 20])
        slice_angle = np.array([305, 300, 300])

# Initiates the variables and arrays used for reading in 'slices' of LiDAR
# data while in various orbits. The width refers to the previously noted
# 'spread'.

        lDrive = np.array([240, 225, 215])

# Initiates the variable and array that determine the speed of the 'left'
# motor for various orbits.

        left_thres = np.array([300, 320, 340])
        right_thres = np.array([305, 325, 345])

# Initiates variables and arrays associated with the minimum and maximum
# distances from the central structure that the robot should remain between
# when orbiting the central structure.

        left_angle = 270
        right_angle = 60

# Initiates variables that determine the central angles of LiDAR data that
# need to be inspected.

        if (end - start) > orbitTime:
            color = colorMethod()
            if color[1] == start_col:
                case = 2
        else:
            req = colorCorner(colorMethod())
            if req == 1:
                case = 2

# Compares the current robot runtime to the time assigned for robot
# operation. If the robot has run for more than 150 seconds and the facing
# color of the central structure matches the start color, the robot is placed
# in the corner-travelling case. Otherwise, if the color of the objects
# collected matches the color of the central structure's facing color, the
# robot is placed in the corner-travelling case. This section of code is
# replicated several times throughout the rest of the code.

```

```

        x = read()
        dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])
        while np.min(dist2) < 900:
            straight()
            x = read()
            dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])

# Reads the distance to the left and slightly forward of the robot, after
# which the robot remains in a loop travelling forward if this distance is
# less than 900 mm, which approximately corresponds to the distance within
# which the central structure should be.

        left_dist = distance_slice(x, left_angle, 15)

# Reads in 'slices' of LiDAR data to the left of the robot.

        if (end - start) > orbitTime:
            color = colorMethod()
            if color[1] == start_col:
                case = 2
        else:
            req = colorCorner(colorMethod())
            if req == 1:
                case = 2

# Replicated time-checking and compared-color code.

        while (np.min(left_dist) < left_thres[orbit_group]) and
(np.min(left_dist) < right_thres[orbit_group]) and (np.min(dist2) < 900):
            x = read()
            left_dist = distance_slice(x, left_angle, 40)
            dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])
            right.Drive(220, 0)
            left.Drive(210, 1)

# The robot is instructed to maintain forward travel with a slight left turn
# so long as it remains within the assigned thresholds and the central
# structure continues to be detected within the inspected 'slice' of LiDAR
# data.

        x = read()
        dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])

# Refreshed the 'slice' of LiDAR data that the central structure ought to be
# within.

```



```

        if (end - start) > orbitTime:
            color = colorMethod()
            if color[1] == start_col:
                case = 2
                print('Going to corner - time running out')
        else:
            req = colorCorner(colorMethod())
            if req == 1:
                case = 2
                print('Going to corner - colors matched')

# Replicated time-checking and compared-color code.

        while (np.min(left_dist) > right_thres[orbit_group]) and
(np.min(left_dist) > left_thres[orbit_group]) and (np.min(dist2) < 900):
            x = read()
            left_dist = distance_slice(x, left_angle, 40)

            dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])
            right.Drive(185, 0)
            left.Drive(1Drive[orbit_group], 1)

# Should the robot find itself outside both its thresholds, it is instructed
# to turn to the left, so it may find itself within its thresholds again.

        if (end - start) > orbitTime:
            color = colorMethod()
            if color[1] == start_col:
                case = 2
        else:
            req = colorCorner(colorMethod())
            if req == 1:
                case = 2

# Replicated time-checking and compared-color code.

        x = read()
        dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])
        while np.min(dist2) > 900:
            x = read()
            dist2 = distance_slice(x, slice_angle[orbit_group],
slice_width[orbit_group])
            right.Stop()
            left.Drive(190, 1)

# Should the robot no longer detect the central structure, it turns to the
# left until it detects the central structure again.

```

```

##### Case 2 #####

    if case == 2:
        c_angle1 = 24
        c_angle2 = 14
        c_width = 4

# Initiates the variables used in detecting the position of a corner.

    x = read()
    c_dist1 = distance_slice(x,c_angle1,c_width)
    c_dist2 = distance_slice(x,c_angle2,c_width)

# Finds the distances from the robot to the walls at the angles that were
# previously initiated.

    right.Drive(200,0)
    left.Stop()

# Instructs the robot to turn to the right. This will be done until it faces
# a corner, after which it will move towards the corner. This process is
# coded later.

    while any(c_dist2 > c_dist1) == True:
        x = read()
        c_dist1 = distance_slice(x,c_angle1,c_width)
        c_dist2 = distance_slice(x,c_angle2,c_width)
    while any(c_dist2 < c_dist1) == True:
        x = read()
        c_dist1 = distance_slice(x,c_angle1,c_width)
        c_dist2 = distance_slice(x,c_angle2,c_width)

# The robot is instructed to continuously take LiDAR measurements until the
# distance at 14° is less than the distance at 24°.

    angle1 = 3
    width1 = angle1 - 1
    angle2 = 360 - angle1

# Initiates the variables used to inspect a 'slice' of LiDAR data directly in
# front of the robot with low spread.

    dist = np.concatenate([distance_slice(x, angle1, width1),
distance_slice(x, angle2, width1) ])
    while np.min(dist) > 300:
        x = read()
        dist = np.concatenate([distance_slice(x, angle1, width1),
distance_slice(x, angle2, width1) ])

# The robot enters a while loop until the distance in front of it is less
# than 300 mm.

```

```

dx = 0.1
dy = np.diff(dist)/dx
diff = x[10,0] - x[(len(x) - 10),0]

# Initiates the variables used to determine whether the robot is directed
# left or right of a corner that it is traveling towards. This includes the
# calculation of discrete differences between neighboring LiDAR distance
# measurements.

    while diff < 0 and (np.min(dist) > 300):
        right.Drive(220,0)
        left.Drive(245,1)
        x = read()
        dist = np.concatenate([distance_slice(x, angle1, width1),
distance_slice(x, angle2, width1) ])
        diff = x[10,0] - x[(len(x) - 10),0]

# If the robot is facing to the right of a corner, it moves more to the left
# to compensate and enable centering on the corner.

    while diff > 0 and (np.min(dist) > 300):
        right.Drive(245,0)
        left.Drive(220,1)
        x = read()
        dist = np.concatenate([distance_slice(x, angle1, width1),
distance_slice(x, angle2, width1) ])
        diff = x[10,0] - x[(len(x) - 10),0]

# If the robot is facing to the left of a corner, it moves more to the right
# to compensate and enable centering on the corner.

    straight()

# The robot then moves straight forward and repeats the loop until it is less
# than 300 mm from a surface in front of it, which is the corner.

    if (end - start) > orbitTime:
        case = 3

# If the robot has run for more than 150 seconds, it is placed in the third
# case, wherein it would end its operation.

    else:
        reverse_dist = np.array([1000, 900, 800])

# Initializes an array of distances that the robot should reverse to,
# depending on which orbit it is in.

        reverse()
        while x[0,0] < reverse_dist[orbit_group] or x[0,0] == 3000 :
            x = read()
            leftZP()

# The robot reverses until it is a set distance from the corner, after which
# it does a zero-point left turn.

```

```

x = read()
angle = 295
dist = distance_slice(x, angle, 5)
while(np.min(dist) < 900):
    x = read()
    dist = distance_slice(x, angle, 5)
while(np.min(dist) > 900):
    x = read()
    dist = distance_slice(x, angle, 5)
case = 1

# The robot continues its zero-point left turn until it detects the presence
# of the central structure around 295°, after which it enters the first case
# again.

    if orbit_group == 2:
        orbit_group = 0

# If the robot is in its outermost orbit group, then it is set back to its
# innermost orbit group.

    if (corner_count % 2) == 0:
        orbit_group = orbit_group + 1

# For every two travels to and returns from a corner the robot completes, the
# orbit group of the robot is incremented by 1.

    corner_count = corner_count + 1

# The corner count of the robot is incremented every time it moves to and
# from a corner.

##### Case 3 #####

    if case == 3:
        stop()
        break

# If the robot has entered its third case, it stops moving and operating, as
# it has run its course.

```