Senior Design
Final Report

IEEE SoutheastCon Hardware Competition

Auburn University

April 29, 2019

Matthew Castleberry, Joe Hinely, Josh Jabłonowski, Nia Perkins

Project and Competition Overview

The purpose of this project is to create a robot to compete in the 2019 IEEE SoutheastCon held in Huntsville, AL. The conference is centered around sharing and demonstrating technical advancements in engineering. We competed in the hardware competition against 40 other universities from across the southeastern US.

The function of the robot is to move around a square carpeted area with a side length of 244 centimeters, and pick up small objects on the field and place them into one of four colored home bases based on the color of the object. The arena is split into two zones. Zone one features four home bases corresponding to four different colors. The robot starts in one of the four home bases. Zone two is separated by a one-inch thick white line. Zone two is where twelve objects are placed for the robot to pick up: eight two inch wooden cubes, and four 2.5 inch diameter pit balls. The arena features a wooden box in the middle that serves as an obstacle as well as four LED lights situated at the edge of zone two on the one-inch line. The four home bases are placed at the four corners of the arena in zone one. An arena was built in order to test the robot as shown in Figure 0. This allows testing of the robot in a controlled area leading up to the competition.

Figure 0: Field Constructed in Broun Hall room 368

The competition had some other ways to score points such as leaving the home base, going in circles (orbits) around the center structure, and raising a flag. There is a penalty to colliding with the flashing lights (spacetels). Table 1 displays a summary of the scoring methods.

Table 1: Summary of scoring methods [1]

Points	Task
5 pts	Leave home base and enter Zone 1
5 pts	Cross the orbital line into Zone 2 (first time only)
5 pts	For each complete, counter-clockwise orbit within Zone 2, starting from the quadrant closest to designated corner square
10 pts	Debris removed from Zone 2 (each)
10 pts	Debris placed in corner square (additional to removal)
10 pts	Color-matched debris placed in appropriate color corner square (bonus points)
10 pts	Finish in your home base
25 pts	At conclusion of debris removal, raise your onboard flag while in home base
-10 pts	Every collision with a Spacetel

The robot has several constraints that limit the size of the robot, materials used in the construction that assure the safety of participants and other robots [1]. The robot's width and length must be within a nine by nine-inch area and a max height of eleven inches. The nine by nine area must include a bumper covering eighty percent of the perimeter. The bumper must be 1.5 inches from the ground and have a minimum vertical cover of one inch, and must not have a radius of curvature less than one centimeter. This is to prevent damage to the arena and other robots in the event of a collision. The robot can also raise a mechanical arm, with a max length of three inches, with a flag presenting the competing school's name.

The robot utilizes two Raspberry Pi microcomputers to control the motors while using an attached camera to identify the color and shape of the objects on the field. A LIDAR is used to localize the robot on the arena to help avoid the four lights and the wooden box in the middle of the arena. The design of the robot is to move the objects to the appropriate home base by running over the objects, closing a small door on the front of the robot, hold them under the robot and move towards the appropriate home base of the object. Figure 1 shows the final robot design.

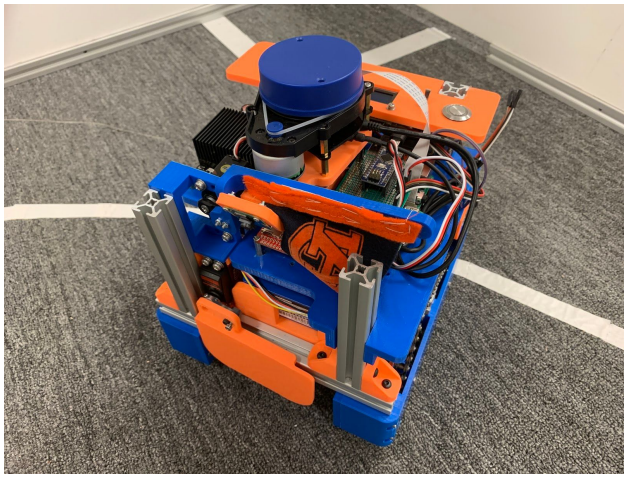

Figure 1: Final robot design

Technical Overview

Mechanical Hardware

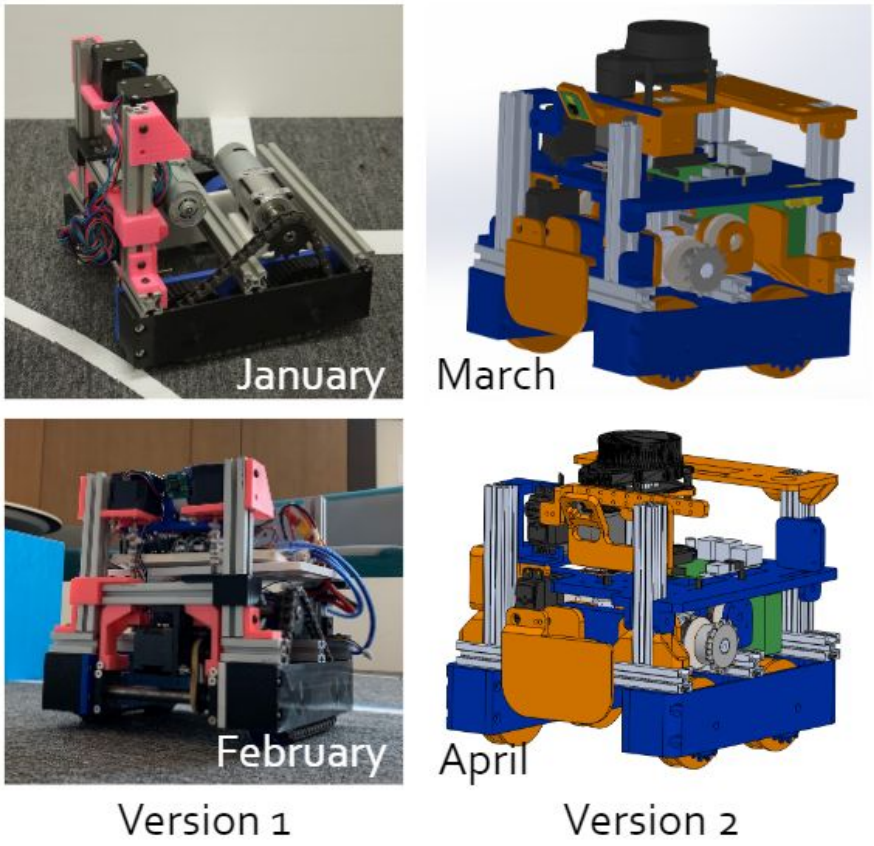

Figure 2: Mechanical Progression

We constructed a 4 wheel differential drive robot. Each side has a motor powering the wheels with a chain and sprocket system. There is an opening with a movable flap in the center section on the bottom for the debris to be collected. The robot fits within the 9” x 9” x 11” size requirement. It was constructed with an 8020 aluminum frame and custom PLA+ 3D printed brackets.

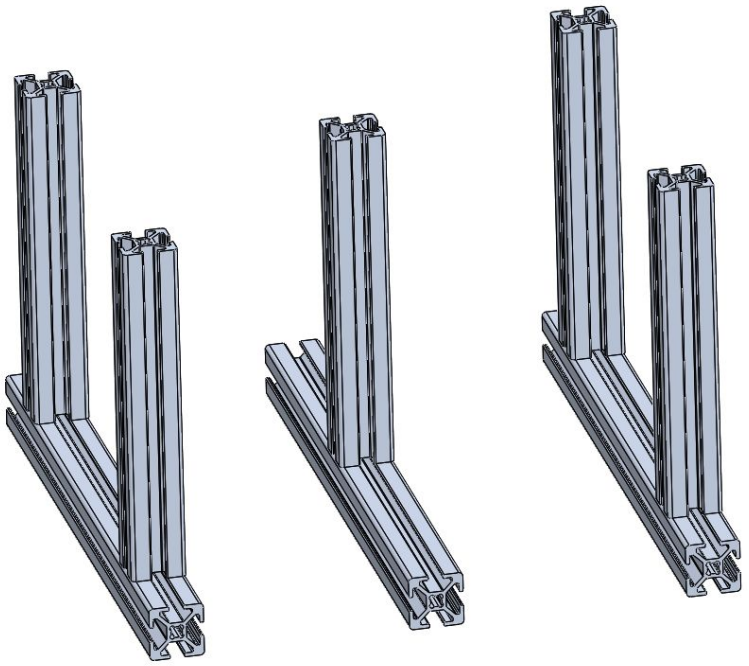

Figure 3: 8020 Frame

We initially intended on using a spinning rod to actuate vertically to collect and dispose of the debris. We called this mechanism the gate. It utilized three stepper motors. During the testing of the robot, the stepper motors were run using a basic program to make them turn. During this test it was discovered that the rubber tubing connecting the stepper motors to the threaded poles was not tight enough, allowing the motors to slip through the tubing. This, alongside the front gate making the robot front heavy lead to the decision to remove the stepper motors and front gate completely. The front gate can be seen in Figure 2 with the pink 3D printed material and the two servo motors on top. In its place, a servo motor was mounted onto the side of the robot and attached to a paddle that will lift it up to allow objects underneath the robot as shown in Figure 1.

The servo motor not only solves the connection problem of the stepper motors but is considerably easier to program. Removing the added weight from the stepper motors at the top of the gate also lowered the center of gravity. This allows the robot to make more sudden stops and maneuver easier around the obstacles on the field. The simpler mechanical approach helped free up more space for additional electronics.

Another issue that was encountered, was the LIDAR needed to be lowered in order to account for the field being constructed in lumber sizes instead of the state height. At the original position, testing proved that the LIDAR would measure some distances beyond the dimensions of the field. A few different parts of the frame and electronics had to be adjusted to account for this. Once this change was made, the LIDAR could not detect anything above the arena walls.

Electrical Hardware

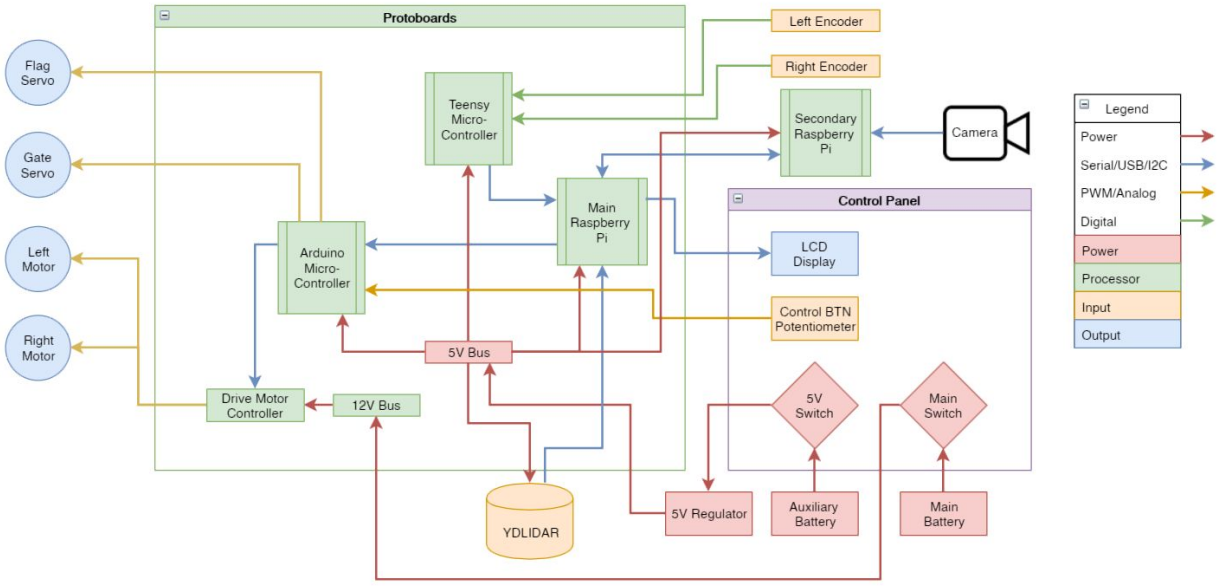

Figure 4: Electrical Block Diagram

We selected 3 different sensors to help the robot be aware of its surroundings in the arena: a visual camera, a 2D LIDAR, and rotary encoders. The camera used is the Raspberry Pi Camera Module. The raw images are processed on entirely on a Raspberry Pi to find the debris objects on the field. The LIDAR is a relatively low-cost 2D LIDAR system. The YDLIDAR uses a spinning laser rangefinder to return a 2D point cloud at about 8 Hz. The rotary encoders monitor the rotation of each drive motor. The LIDAR and encoders are used for localizing the robot in the field. We originally were going to utilize an inertial measurement unit (IMU) to help receive information on orientation and acceleration but we did not use it in the final design.

The robot is controlled using two coprocessing Raspberry Pi 3B+ connected via an ethernet cable. A Teensy microcontroller polls the encoders and sends the velocity information to the main Pi. Once the Pi's determine the desired path, an Arduino microcontroller instructs the motor controllers of the corresponding commands. Two different batteries are used to prevent the motors from causing a brownout situation for the Raspberry Pi's. In Figure 4, the latest version of the electrical block diagram shows all of the components on the robot.

To facilitate rapid prototyping, protoboards were used to keep most of the main components centralized to one area and clean up some of the wiring. Two primary peripherals were used to supplement the Raspberry Pi's as I/O: an Arduino Nano on the main protoboard and a Teensy on its own protoboard. Figure 5 shows a wiring diagram of how the protoboards were utilized.

The Arduino Nano served to wire most of the components together. This includes the servos, buttons, LCD, selector switch, and main motor controller. The decision behind this would be to relieve the processing load from the already over-taxed Raspberry Pi.

The Teensy was required later on in the design process due to the Arduino's lack of available pins and hardware interrupts to properly implement the encoders. It counts the number of ticks the encoders have counted and sends the values via serial bus to the Raspberry Pi. The Pi can then compute this easily into motor distance traveled.

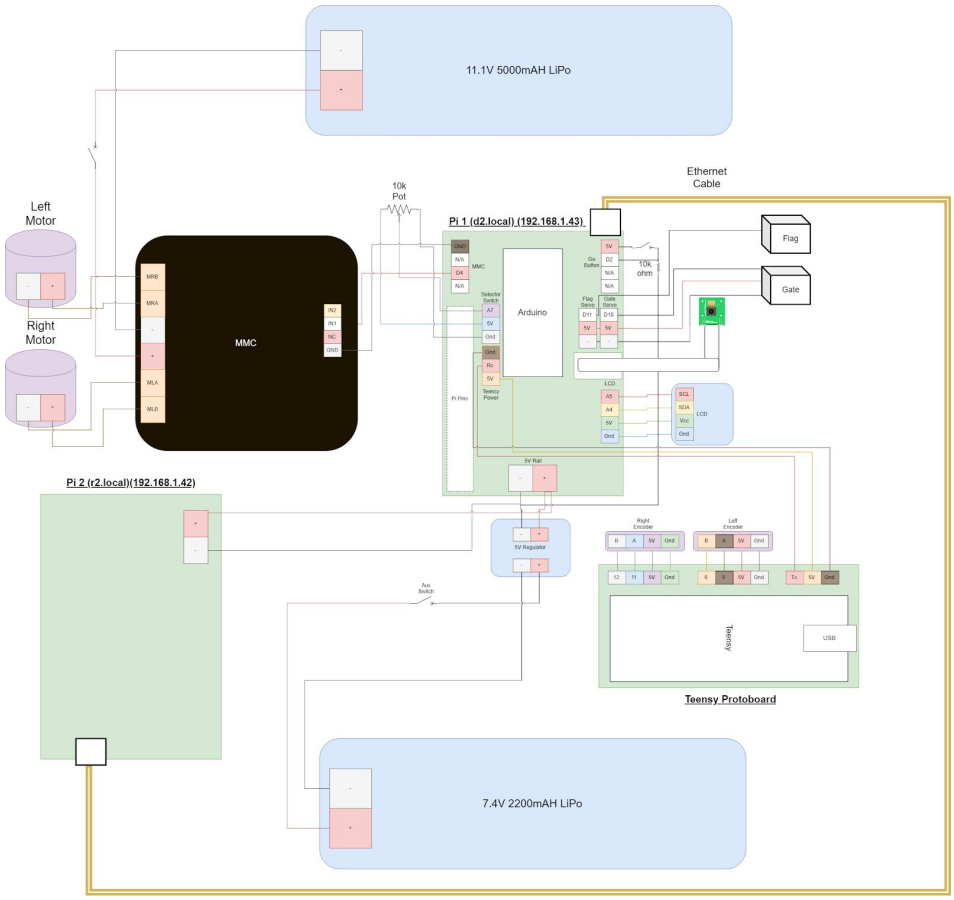

Figure 5: Wiring Diagram

Software

Robot Operating System (ROS)

The need for a simplified approach for localization sparked the switch to using the Robot Operating System (ROS.) ROS greatly simplifies complex localization, mapping, and pathfinding algorithms; all of which are incredibly essential to this competition, although, heed was taken at first because of the vast learning curve coupled with ROS. A very layman's explanation of ROS is that sensor data is input nodes, which house all the vital information for

the sensors such as programs, readme files, or launch files. Nodes can talk to each other and request information, process the information, and send to another node. A graphical explanation of how our sensor nodes communicate is shown in Figure 6 below. We have four sensors; LIDAR, Camera, IMU, and encoders that create four ROS nodes. The LIDAR, IMU, and Encoder nodes all input into the localization node, this node coupled with the camera node create what we call the decision maker or essentially a path planner. The decision maker then outputs motor control commands to move the robot.

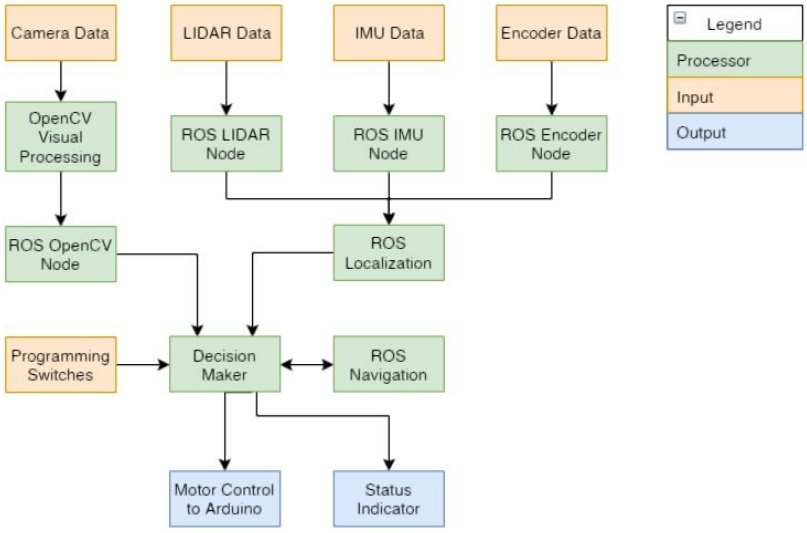

Figure 6: ROS Node Diagram

Figure 7 shows the resulting ROS graph of our implementation. After testing, we determined the IMU was not adding any additional helpful information for localization so it was omitted from the final design.

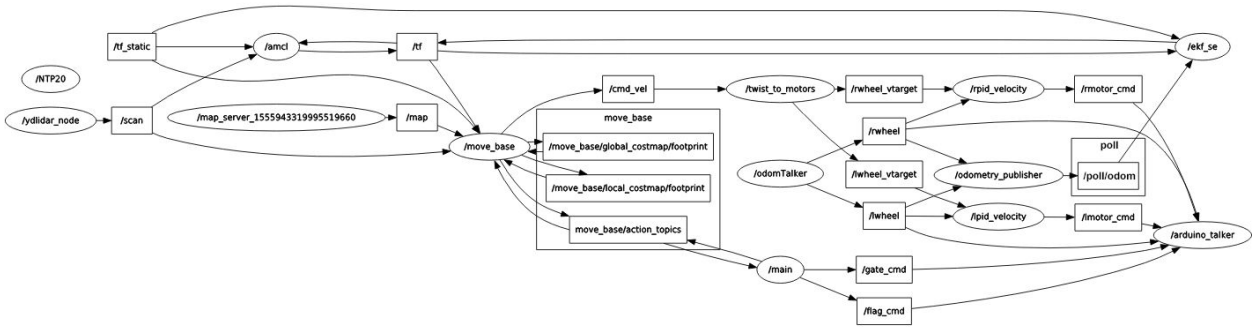

Figure 7: RQT Graph

LIDAR and Localization

Figure 8: YDLIDAR

The LIDAR sensor is one of the main components of localization; this sensor acts sort of like a GPS for the robot. The YDLIDAR (Figure 8) has a 360-degree scanning range with a 10-meter range. As stated previously and shown in Figure 7 the LIDAR will be a node, and this is one of two inputs into the localization node. ROS greatly simplifies the localization process as it already has the algorithms needed, one being AMCL. Adaptive Monte Carlo Localization is the process of tracking the pose of a robot against a known map, which is our case. AMCL takes in a map, LIDAR scan, and transform messages, and outputs pose estimates. As stated one of the first things we need to implement the AMCL algorithm is a known map, which is the competition field. Our original process of creating a map was creating a jpeg image and feeding it thru a python program launched within ROS to return a map. However, errors were given when adding the LIDAR scan data to the map. To solve this issue, we had to pre-map the field with actual data using simultaneous location and mapping (SLAM). This proved to be very effective, especially when there was a large skew in the field shape. See Figure 9 for a view of the robot's orientation and position plotted on a map in ROS.

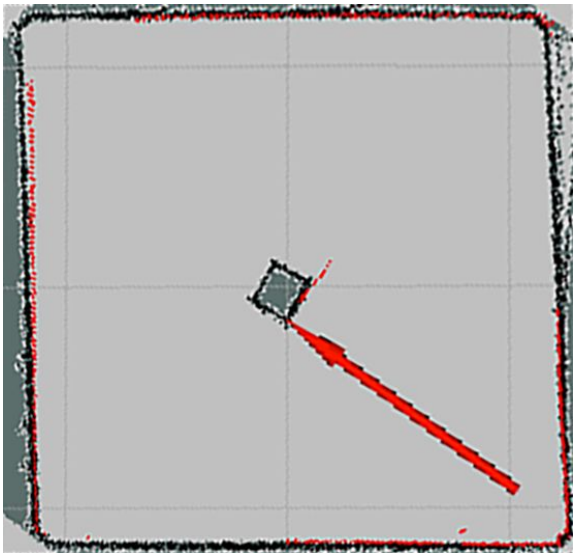

Figure 9: RVIZ map with localized position and orientation

Rotary Encoder and Odometry

Figure 10: Quadrature Encoders

In order to have more precise measurements in movement, we will use two capacitive encoders attached to the motors. The encoders measure and count the turns of the motors, allowing us to know how far the robot will travel with one motor turn. Using this alongside the localization and object detection, we would be able to move the robot the exact distance towards any detected object to ensure the robot picks up the object. This will also prevent the robot from pushing any object into a position that would make it difficult to pick up.

The encoders are wired and read using a Teensy microcontroller. The Teensy features built in interrupts that allow regular updates from the encoders. This is required as the encoders measure rotations in ticks, these ticks update at a very fast frequency, requiring processing just as quickly. The Teensy reads the encoder ticks and calculates the angular acceleration, which is then sent to the Raspberry Pi via serial.

Visual Detection

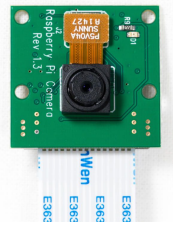

Figure 11: Raspberry Pi Camera Module

We are using the Raspberry Pi Camera Module (Figure 11) to detect the debris objects. By using the open-source computer vision library, the Pi can extrapolate the debris's pixel location, pixel size, the angle from center, color, type, and approximate distance from the robot. It uses this data to make decisions on where the robot should go. For E-Day, we demonstrated that the robot can turn towards an object, drive until the object is right in front of it and then turn and find a different object. The program worked really well. The biggest issue is that the camera was pointed too far up and needed to be mounted at a lower angle. The next step was integrating visual detection with ROS. A custom message group was created to pass the object information to the main node for navigation.

Navigation

Once the robot is capable of finding itself on a map with localization, it becomes necessary to plan the steps from point A to point B. There are several ROS nodes available for this that we attempted to use listed below.

- DWA (Dynamic Window Approach)
    - Smoother paths
    - Requires more computation power
- FTC (Follow the Carrot)
    - More simplistic approach, less computation
    - Does not avoid obstacles well
- Move_Basic
    - Simplest planner
    - Turn to goal, move until goal is reached
    - Not always accurate
    - Does not avoid obstacles

Ultimately, our design settled on Move_Basic, as it provided the simplest approach with parameters that were in our scope of design. Aside from Move_Basic, most path planning algorithms use a form of costmaps. These costmaps determine what path would "cost" the least

in terms of time and distance. Most algorithms also use both a local costmap and a global costmap.

The local costmap will calculate where the robot should go in its immediate vicinity, this is usually calculated in real time as the range of the map is shorter and in relation the position of the robot. The global costmap will calculate a path using algorithms such as A* and output something like what is seen in Figure 12. Using both of these costmaps, the robot can efficiently navigate its space avoiding obstacles it can see.

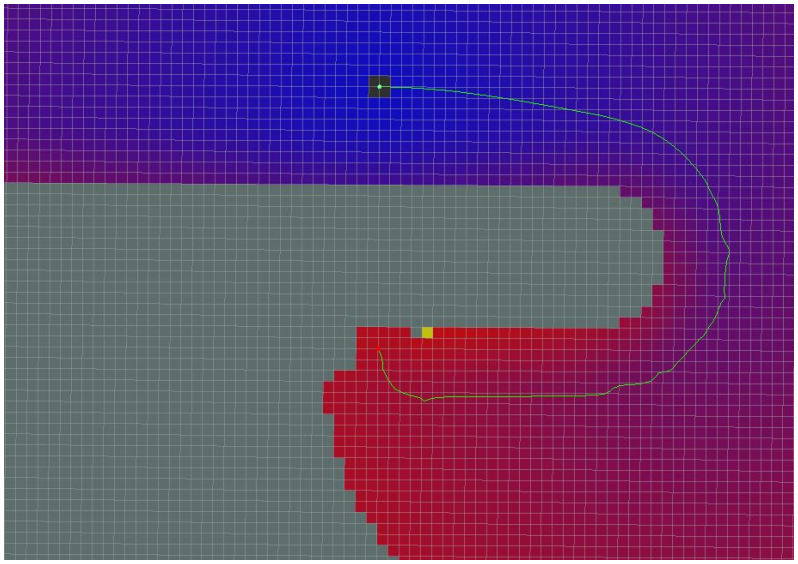

Figure 12: Example visualization of navigational pathfinding [2]

Budget

Table 2: Budget Overview

Category	**Estimated Cost for 1 Robot**	**Actual Cost for 1 Robot**	**Total Expenditures**
Electronics (Pi, Sensors, Controllers, etc.)	$170	$253.63	$487.26
Electrical Hardware (Batteries, wires, etc.)	$75	$84.33	$84.33
Electromechanical (Motors, servos, etc.)	$123.47	$267.39	$316.80
Mechanical (Aluminum, 3D printing filament, etc.)	$345.91	$368.05	$697.14
Total	$714.38	$973.4	$1585.53

Conclusion

While each individual aspect operated nicely when configured properly, the Raspberry Pi's were not computationally powerful enough to handle the load. ROS and OpenCV are very processing intensive. Adding a second Pi helped distribute the load but increased development time dramatically as well as reduced consistency. In order to properly accomplish the tasks with this approach, a more powerful processor is required such as an Intel NUC.

Our robot was the most complex from an electrical and software perspective at the competition. Despite the lack of success at the competition, we learned a lot. Most of the key aspects of the robot worked very well just not simultaneously. An improvement in hardware could dramatically improve performance.

References

[1] "2019 SoutheastCon Hardware Rules v1.3." IEEE, Huntsville, 18-Aug-2018. "http://sites.ieee.org/southeastcon2019/files/2019/02/2019-SoutheastCon-HW-Rules_v1_3-1.pdf "

[2] "ROS Global Path Planner." ROS, 24-Apr-2019. "http://wiki.ros.org/global_planner"