Roborodentia XXI Robot

Jose A. Villa
Advisor: John Seng
California Polytechnic State University
Computer Engineering Program
Spring 2016

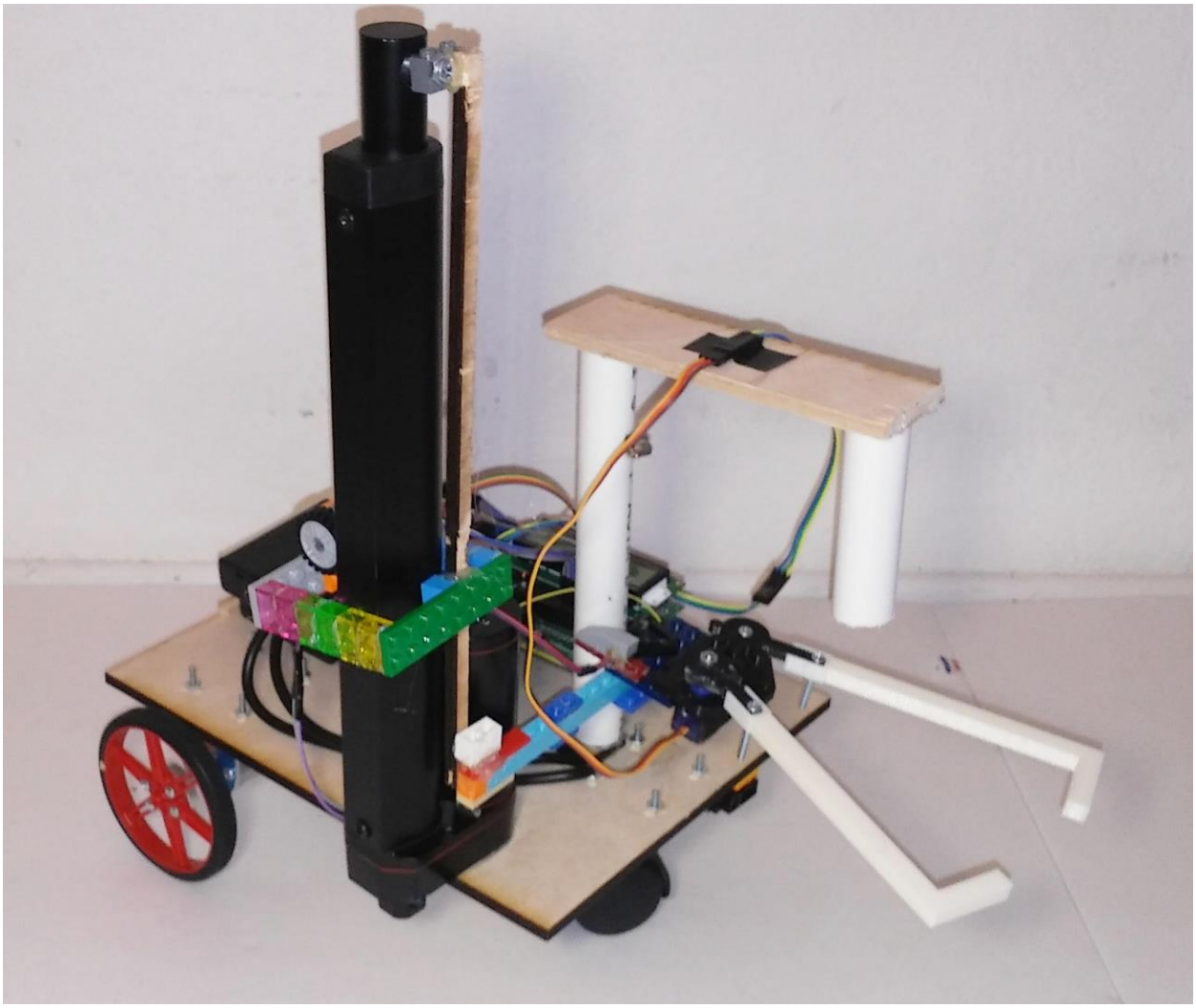

Table of Contents

Table of Figures

Introduction

Roborodentia is a competition held annually during Cal Poly's Open House. The general objective of every competition is to build an autonomous robot that will perform specific tasks to earn points within a time limit. These tasks are different every year and sometimes are radically changed from the previous competition. Building such robots involves a good mix of mechanical, electrical, and software knowledge. The project outlined in this report was a contestant robot for the 21st Roborodentia competition which took place on April 16, 2016.

Problem Statement

Overview

The overall objective of the competition was for the robot to carry rings from a supply peg to a scoring peg. Figure 1 shows a model of the course. Robots can take primary, secondary, or center rings and score points by placing them on scoring pegs #1 or #2. Below is a high level summary of the contest rules. Full specifications can be found in Appendix A [1].

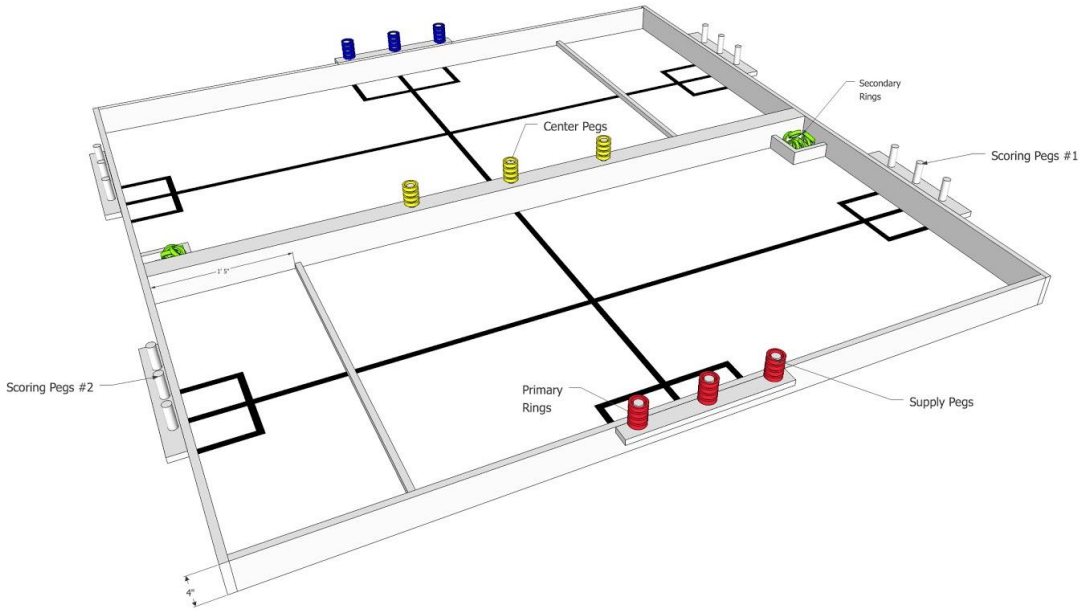

Figure 1. Roborodentia XXI Course Model, Perspective View

Robot Specifications

Robot footprint must be 12”x12” maximum at the beginning of the match and must not exceed 14”x14” at any point during the match. Height must be under 15” at the beginning of the match with no restriction once the match begins. Robots are to be fully autonomous and may not fly or disassemble into multiple parts.

Course Specifications

The entire course is 8’x8’, with 4” walls surrounding the edges and along the center. The black lines (shown in **Error! Reference source not found.** and Figure 2) are strips of ¾” black masking tape. Rings are cut from 2” Sch 40 PVC pipes and are ½” tall. All the pegs are cut from ½” Sch 40 PVC pipes and are 3” tall. Each supply and center peg holds 4 rings. The secondary ring box holds 10 secondary rings.

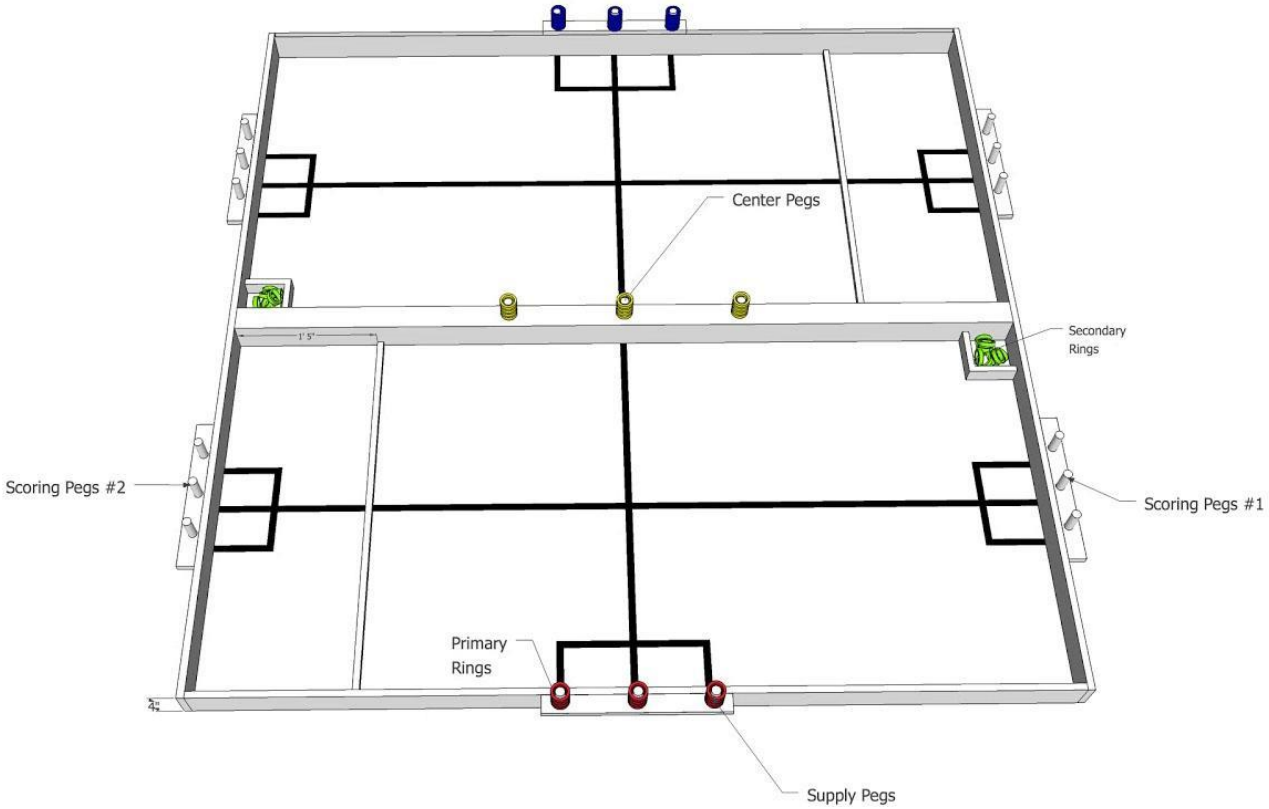

Figure 2. Roborodentia XXI Course Model, Top View

Regulations

The competition was a head-to-head double elimination tournament with 3-minute matches. Competitors were seeded based on qualifying runs. At the beginning of the match, robots must be touching the tape intersection closest to the supply pegs. Each supply peg will be replenished with up to 4 rings every time the robot touches the center intersection; center pegs are not replenished throughout the match.

Scoring

Every primary ring placed in a scoring peg #1 is worth 1 point and 3 points if it is placed in a scoring peg #2. Secondary rings are worth 2 points if placed in scoring peg #1 and 8 points if placed in scoring peg #2. Center rings are worth 3 points on their own; moreover, when a center ring is part of a stack of rings on a scoring peg, the value of the stack is tripled but the center ring only counts as a primary ring. When a scoring peg holds 4 or more rings, they will be removed and only the bottom 4 rings will score points.

Design

Software

Overview

The software that runs the system is written for the Arduino platform which uses an avr-gcc compiler on C/C++ code that is later executed by the ATmega2560. The entirety of the code (not including standard libraries) is contained in four files: RoboShield.cpp, RoboShield.h, RoboShield_Defines.h, and RoboShieldMain.ino. The first three files comprise the library provided for the Roboshield, while the last one contains all functionality written specifically for the competition. The Roboshield library can be found on github.com [2] and RoboShield.ino can be found in Appendix B. Below are detailed descriptions of each.

RoboShield Library

This library was written specifically for the RoboShield to run on the ATmega2560. It was developed by Cal Poly professor Dr. John Seng and Cal Poly alumnus Brian Gomberg, both of who also designed the RoboShield board itself. The library is split into three files and consists of a single class called RoboShield which contains about thirty methods. RoboShield.h contains the class and function declarations, RoboShield_Defines.h contains all the #defines used throughout the library, and RoboShield.cpp contains the class and function definitions.

RoboShieldMain.ino

This file contains the main loop performed for the competition as well as some support functions. The main loop begins by picking up rings from the supply pegs, which is where the robot begins the match. After that, the robot backs up 6”, turns around, and heads towards the center of the course as it follows the black line. Once it sees the intersection, it takes a right turn and continues to follow the black line as it heads towards the scoring pegs. Once it sees the intersection by the scoring pegs, it stops and begins to move forward until the distance sensor reads about 1” away from the wall. At that point, the rings are lowered onto the scoring pegs. The robot once again backs up for 6”, turns around and follows the black line until the center intersection. It then takes a left turn, stops at the intersection by the supply pegs, and gets within 1” of the wall to begin the loop again. Figure 3 graphically shows the process in a flowchart.

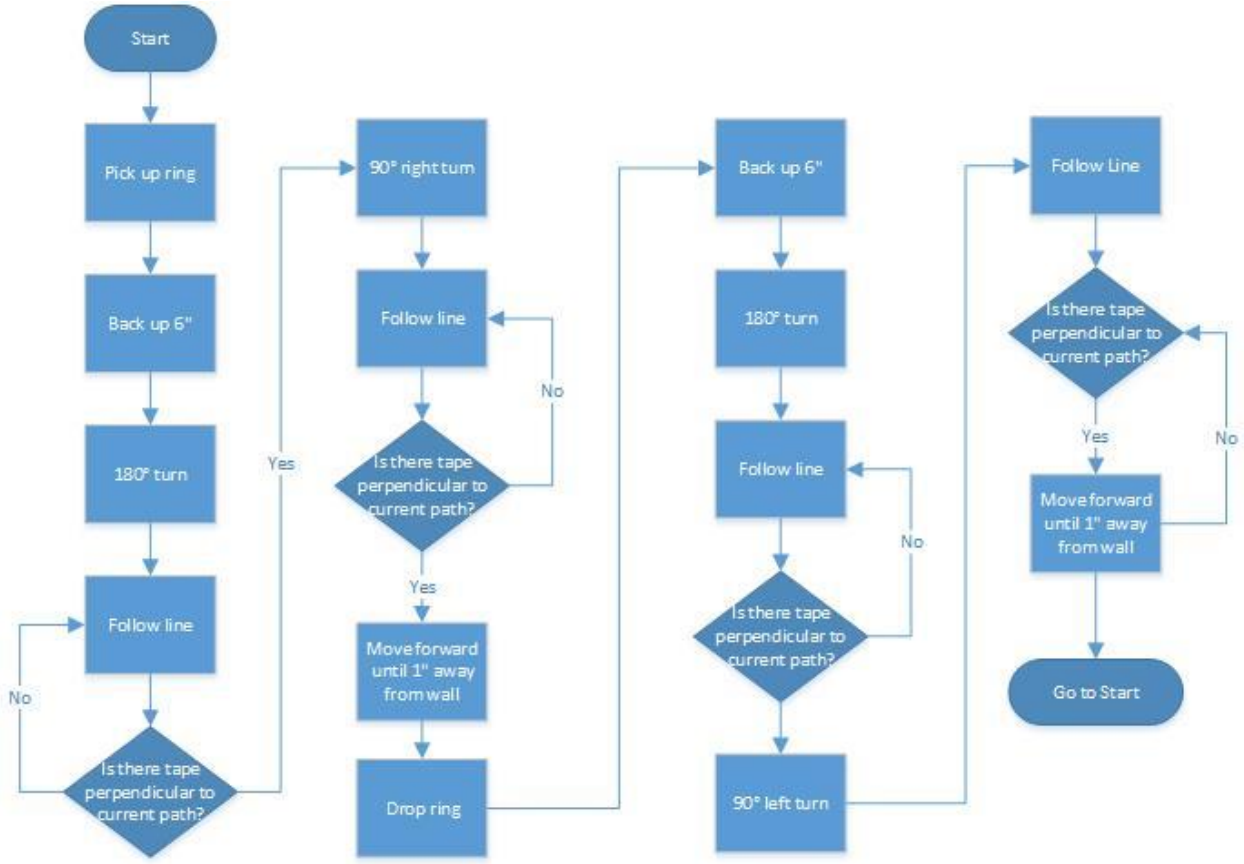

Figure 3. High level program flow

The rest of the RoboShieldMain.ino file contains functions that support each of the processes outlined in the flowchart. Every process can be divided into four different categories. Below are detailed descriptions of each.

Pick Up/Drop Ring

This process is performed by the LiftRing() function, shown in Code Snippet 1 (note that 'shield' is a global reference to a RoboShield object). A detailed mechanical description of this process can be found in the Mechanical subsection. It begins by setting the angle of the mini servo so that the gripper that is attached to it closes enough to hold a ring (the gripper should initially be opened). This angle was fine-tuned by trial and error. Two magnets and the reed switch are used to signal that actuator is low enough to pick up a ring or high enough to clear a peg. Since the actuator begins in the low position, the function tells the actuator to begin moving up for 5000ms, which is long enough for the reed switch to clear the bottom magnet. At that point, it begins waiting for the reed switch to see the top magnet and stop the actuator.

Code Snippet 1: LiftRing()function.

```
void LiftRing () {
   shield.setServo(GRIPPER_PIN, GRIPPER_CLOSED);    //Close gripper
   shield.setMotor(ACTUATOR_PIN, ACTUATOR_UP);      //Begin moving the linear actuator up
   delay(5000);                                     //Wait for the actuator to clear the bottom magnet
   while (digitalRead(REED_SENSOR_PIN) == 0)        //Wait for the reed switch to see the top magnet
      ;
   shield.setMotor(ACTUATOR_PIN, 0);                //Stop the actuator
}
```

The LowerRing() function performs the same calls but with different arguments and in different order. It initially instructs the linear actuator to move down and opens the gripper after the bottom magnet is sensed by the reed switch.

Distance-specific maneuvers

A mathematical approach was taken to approximate distances to for the wheels to travel. Each gearmotor has an integrated quadrature encoder that outputs 64 CPR (counts per revolution) of the motor shaft. The gear ratio of the output shaft to motor shaft is 50:1, which totals (50)(64 CPR) = 3200 CPR of the output shaft. However, the library only looks at one of the two encoder outputs, and only counts rising edges of the signal effectively cutting the amount of CPRs to a fourth. The final count is then 3200 CPR / 4 = 800 CPR. The wheels used were 90mm in diameter and therefore (90mm)(π) ≈ 283mm in circumference. So the encoders give a distance per count of (283mm / 800 counts) = 0.35375mm/count or 0.014”/count. Simple conversions can be done to get the counts needed for a wheel to travel a specific distance. For example, one of the processes requires the robot to back up 6”, which comes out to be (6”) / (0.014”/count) ≈ 429 counts.

A little more math was required to calculate turns. Assuming only one of the wheels moves and the other one rotates in place, and approximation of the distance that it takes for one of the wheel to travel to complete a specific turn in degrees can be calculated using the formula for the circumference of a circle and taking the distance between the two wheels as the radius. Figure 4 illustrates the idea and shows an example for calculating a 90° turn.

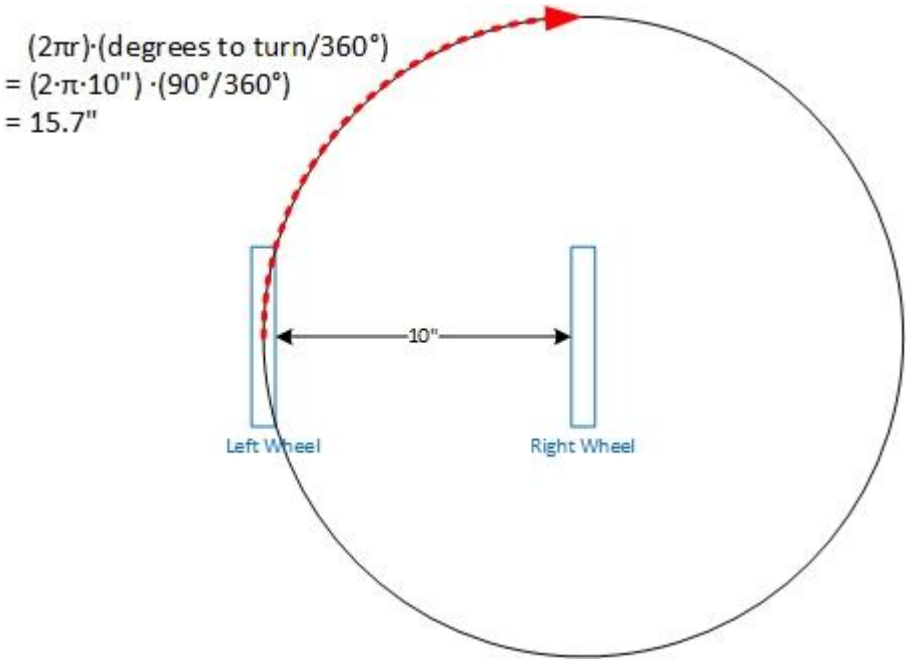

Figure 4. 90° Turn Calculation

From there, we can use the same formula as before and get the number of encoder counts needed for the left wheel to travel that distance. Specifically, (15.7”) / (0.014”/count) ≈ 1121 counts. Specific values for the needed distances were computed beforehand and stored as #define constants to avoid computation costs.

Moving to a specific distance from the wall

This is a straightforward function that simply moves the robot forward until the value being read from the IR distance sensor exceeds a specific value. It is worth noting, however, that a specific distance cannot be accurately calculated from a sensor reading but rather has to be extracted from the sensor’s datasheet as the distance-voltage curve is not exactly linear. The only distance used is 1”, which the datasheet says should result in a sensor output of about 1.9 volts [3]. The microcontroller’s Analog to Digital Converter (ADC) takes a 0V-5V reading and maps it to a 0-1023 integer output. This means the 1” should roughly be (1.9V/5V)·(1023) ≈ 389, which is argument used in the function call AdvanceUntilIR(), shown in Code Snippet 2.

Code Snippet 2: AdvanceUntilIR()function.

```
void advanceUntilIR(int value) {
    //Set motors to move forward
    shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED);
    shield.setMotor(RIGHT_MOTOR_PIN, -BASE_SPEED);

    //Wait for specific distance
    while (shield.getAnalog(IR_SENSOR_PIN) < value)
        ;

    //Stop motors
    shield.setMotor(LEFT_MOTOR_PIN, 0);
    shield.setMotor(RIGHT_MOTOR_PIN, 0);
}
```

Line following and stopping

This was the section that required the most time and testing. With a variety of line following algorithms and an array of eight reflectance sensors to take advantage of, options were vast. Figure 5 shows the algorithm that was used. Only the middle four sensors, out of the eight, were used. The sensors will be referred to as Outer Left (OL), Center Left (CL), Center Right (CR), and Outer Right (OR) hereafter. The algorithm works under the assumption that at the time that it's called, there is a black line under at least one of the four sensors. The algorithm begins by taking a reading from the four sensors. Six different scenarios were considered and are illustrated in Figure 6.

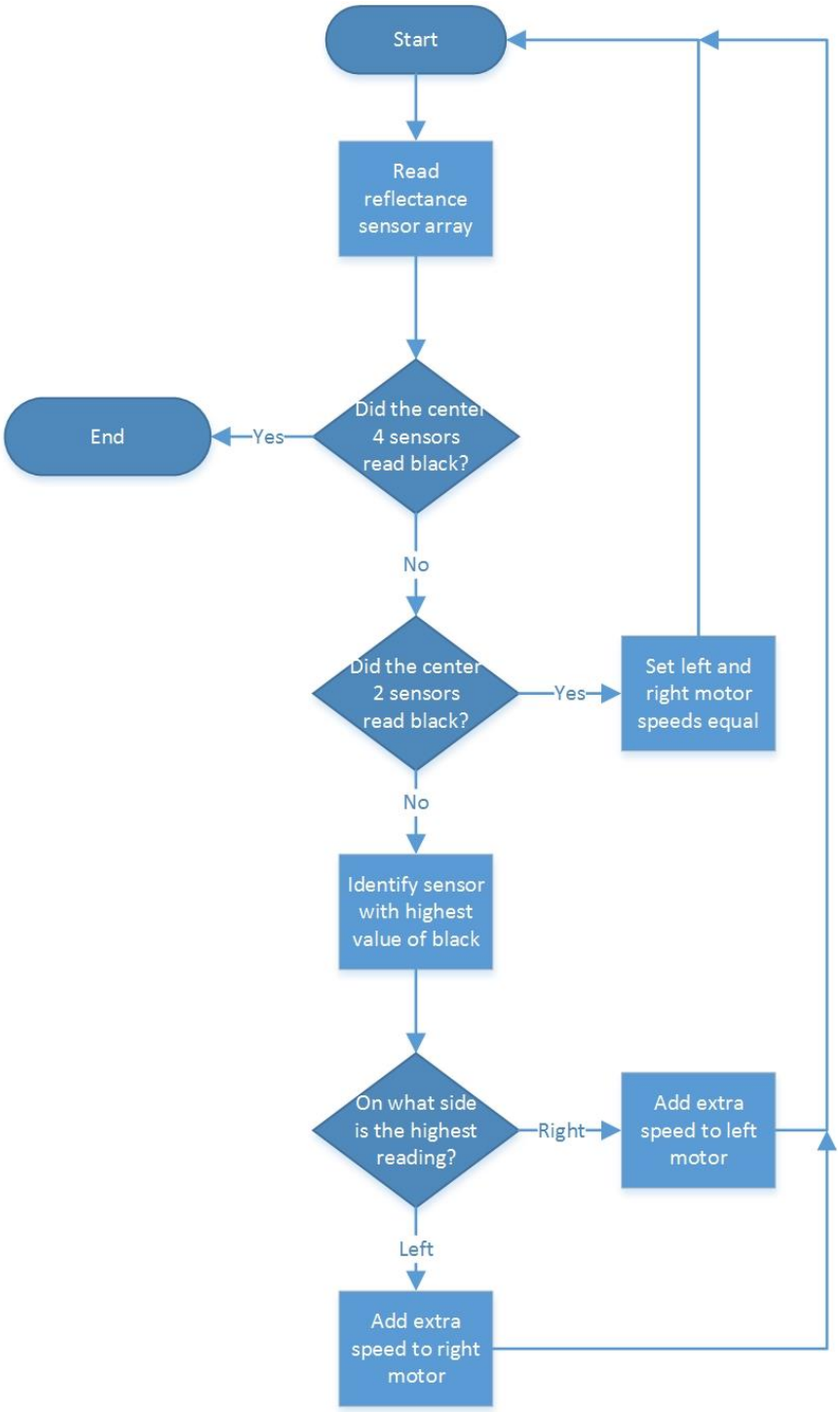

Figure 5. Line Following Algorithm Flowchart

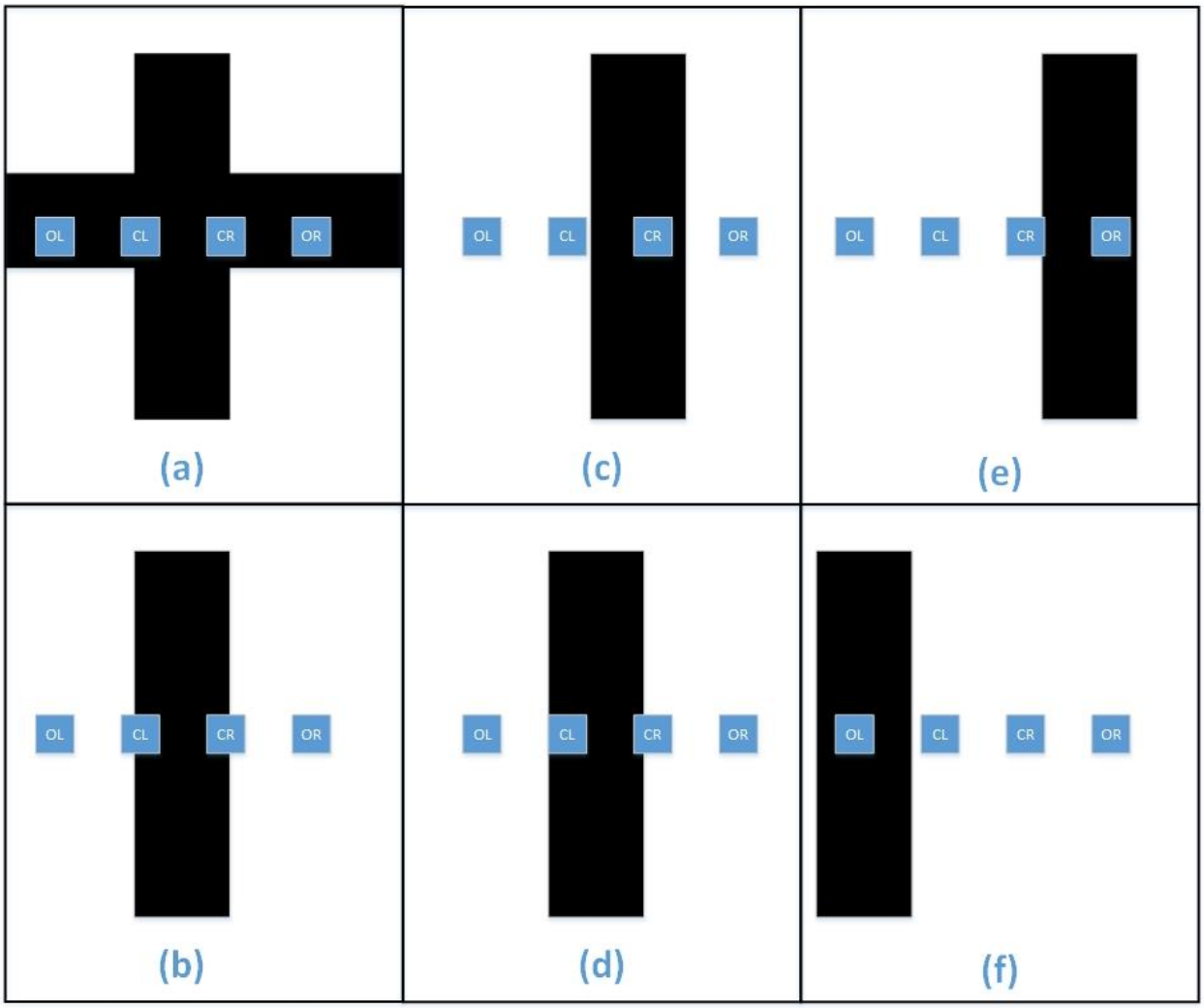

Figure 6. Line Following Algorithm Scenarios

Scenario (a) is the one that is first checked for. It is whether or not the robot found a horizontal black tape line. This immediately stops both motors and ends the algorithm. In scenario (b) both center sensors are able to read the line. In practice, this is an 'if statement' checking that both CL and CR sensor readings are above a specific threshold. This is the ideal scenario as the robot is presumably following the black line in a straight path. Both motor speeds are set to the same value expecting to maintain the same path and the algorithm starts over. In case at least either CL or CR read less than the threshold, the next step is to find the max value among the sensor readings, i.e. which sensor is closest to the black line. This is a simple search for the highest value on a four-element integer array with the sensor values. In case of a tie, the first found value is taken as the highest. This leaves the last four scenarios: scenario (c) where CR is the max value, scenario (d) where CL is the max value, scenario (e) where OR is the max value, and scenario (f) where OL is the max value. Note on scenario (d) that the value of sensor CR would not be above the threshold for a black line and is therefore different from scenario (a). Scenarios (c) and (e) result in speeding up the left motor and

scenarios (d) and (f) in speeding up the right motor, as pointed out in the algorithm flowchart. The amount of difference in speed, or correction, is determined by both the value of the max sensor reading, and whether it is a center or outer sensor that has the max value. Higher max sensor readings and outer sensors result in higher correction. Finally, there is a small delay for allowing the correction to impact the path before returning to the beginning of the algorithm and recalculating correction.

Hardware

Overview

The system was controlled by an ATmega2560 on an Arduino Mega-clone microcontroller board, manufactured by Keystudio. On top of it sat a Roboshield, developed by Cal Poly professor Dr. John Seng and Cal Poly alumnus Brian Gomberg. Wired to the microcontroller/shield were two gearmotors, a reed switch, a linear actuator, a mini servo, an IR distance sensor, and a reflectance sensor array. The system was powered by 8 AA-rechargeable-batteries. Figure 7 shows a high level diagram of the system. There were two main subsystems: the navigation and the ring pick-up/drop-off.

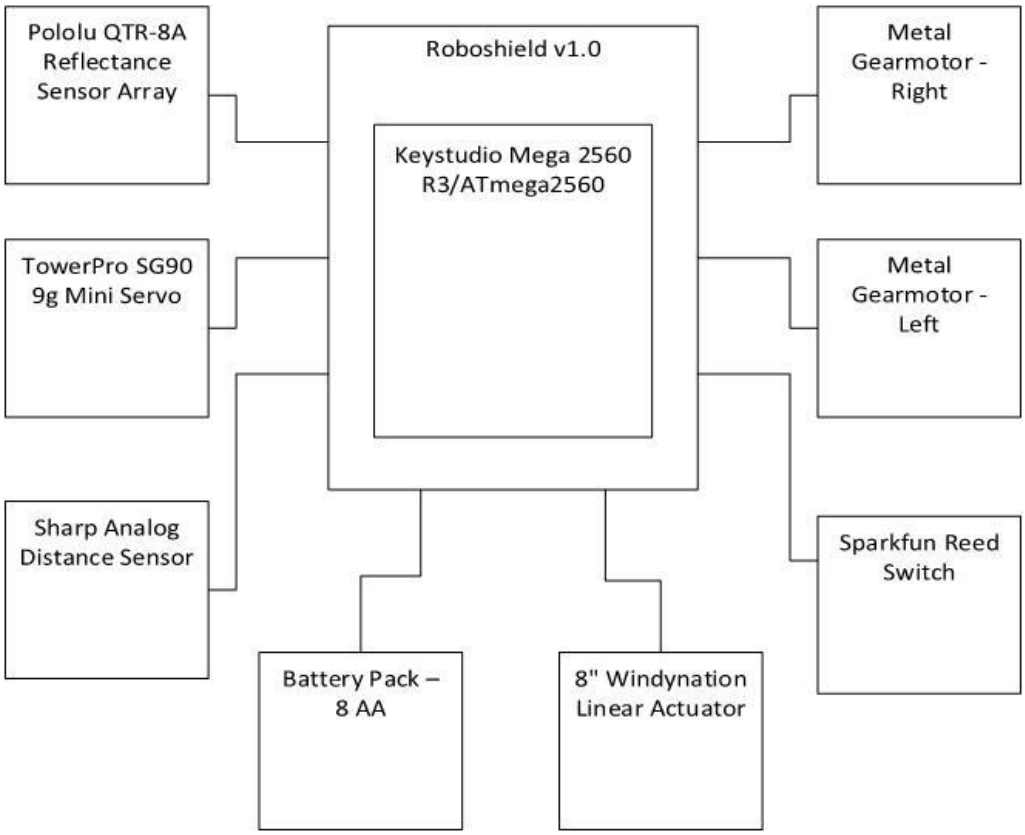

Figure 7. High Level Architecture Diagram

Navigation

Movement was achieved by rotating two Pololu 50:1 Metal Gearmotors with 64 CPR encoders. Each was connected to the two 6-pin motor connectors in the Roboshield which include motor terminal A, motor terminal B, encoder GND, encoder Vcc, encoder A output, and encoder B output. Line following was achieved with the aid of a Pololu QTR-8A Reflectance Sensor Array. The sensor provides analog output readings from each of the 8 IR LED/phototransistor pairs, these were captured by input pins A0 through A7 on the Roboshield. Finally, proximity to walls was measured by a Sharp GP2Y0A51SK0F Analog Distance Sensor with a range of 2-15cm. Figure 8 shows the wiring diagram.

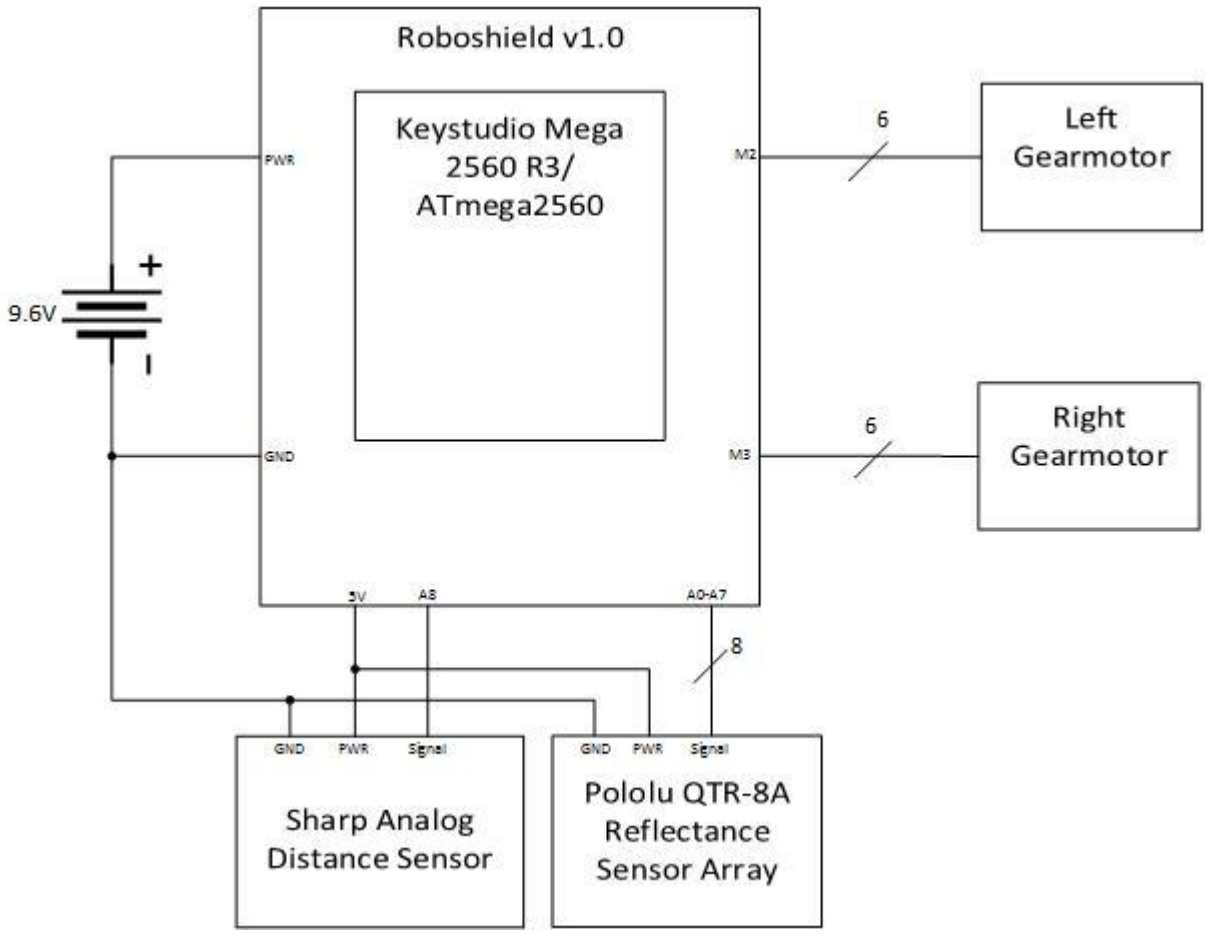

Figure 8. Navigation Subsystem Diagram

Ring Pick-up/Drop-off

This subsystem included a PowerPro SG90 9g Mini Servo which actuated a 3D printed gripper. The three pin servo connector (ground, power, and signal) plugged right into one of the eight servo connectors on the Roboshield. The gripper was attached to the Windynation Linear Actuator so that the rings could be lifted. The linear actuator was controlled similarly to a DC motor and is therefore able to be connected to one of the four two-pin motor connectors on the Roboshield. Finally, the reed switch and two magnets provide information regarding the position of the linear actuator. The reed switch acted as a short when a magnet was nearby and otherwise as an open. It was therefore wired as a simple switch and the value was read through a digital pin in the Keystudio microcontroller. One of the magnets signaled the gripper was low enough to grab rings and the other one that the gripper was high enough to clear the peg. Figure 9 shows the wiring diagram.

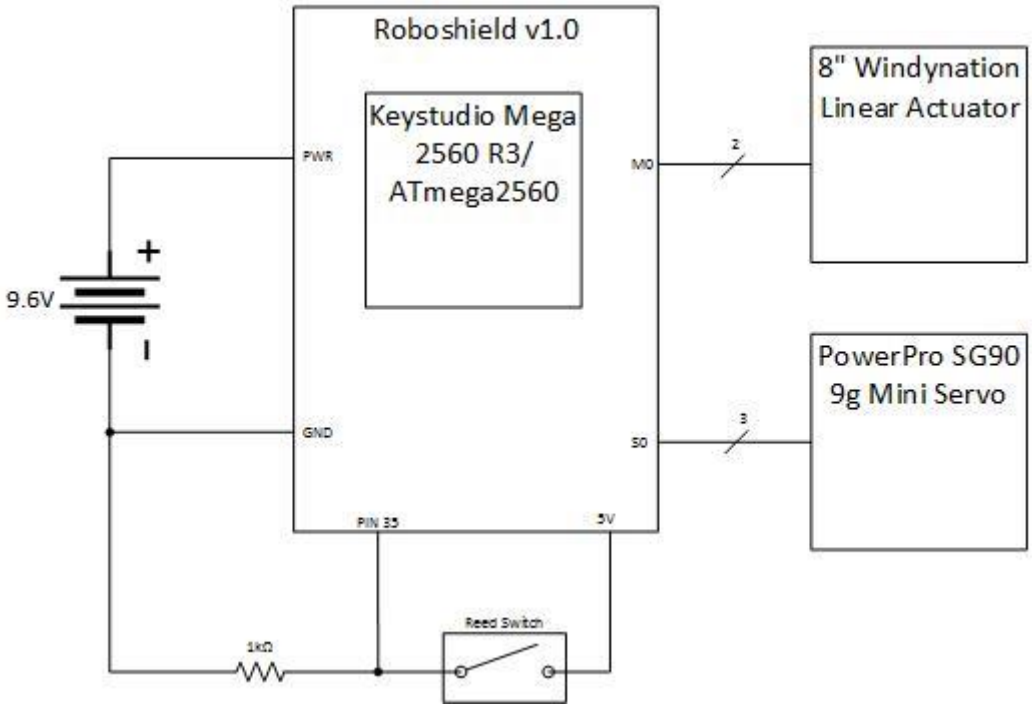

Figure 9. Ring Pick-up Subsystem Diagram

Mechanical

Overview

The base of the robot was a laser-cut wood chassis designed using Autodesk Inventor. The chassis had four screw holes for mounting each gearmotor, four screw holes for the front swivel wheel, four screw holes to mount the Keystudio microcontroller, and a squared opening to run wires from the bottom to the top of the chassis. Figure 10 shows the model.

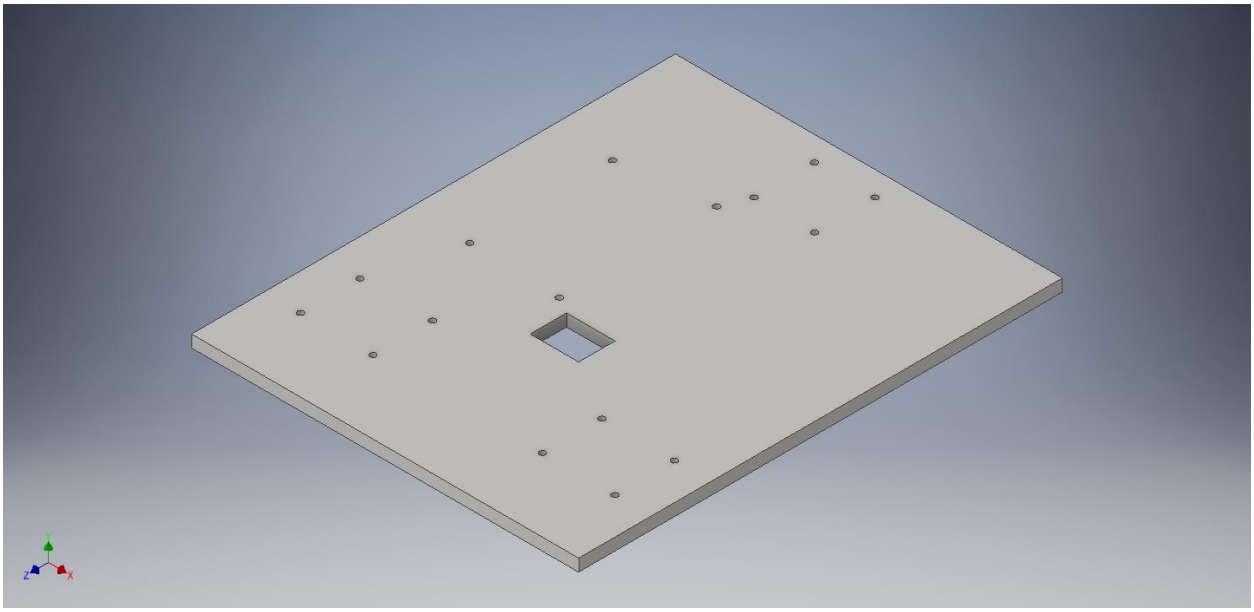

Figure 10. Chassis CAD Model.

On the bottom side of the chassis, there were two mounting brackets for the gearmotors attached using four screws each. Close to the center of the chassis was the reflectance sensor array mounted using LEGO blocks and hot-melt adhesive. Near the front was a swivel wheel attached to the four mounting screw holes. Next to it was the distance sensor hot-glued to the chassis. Figure 11 shows a pictorial representation.

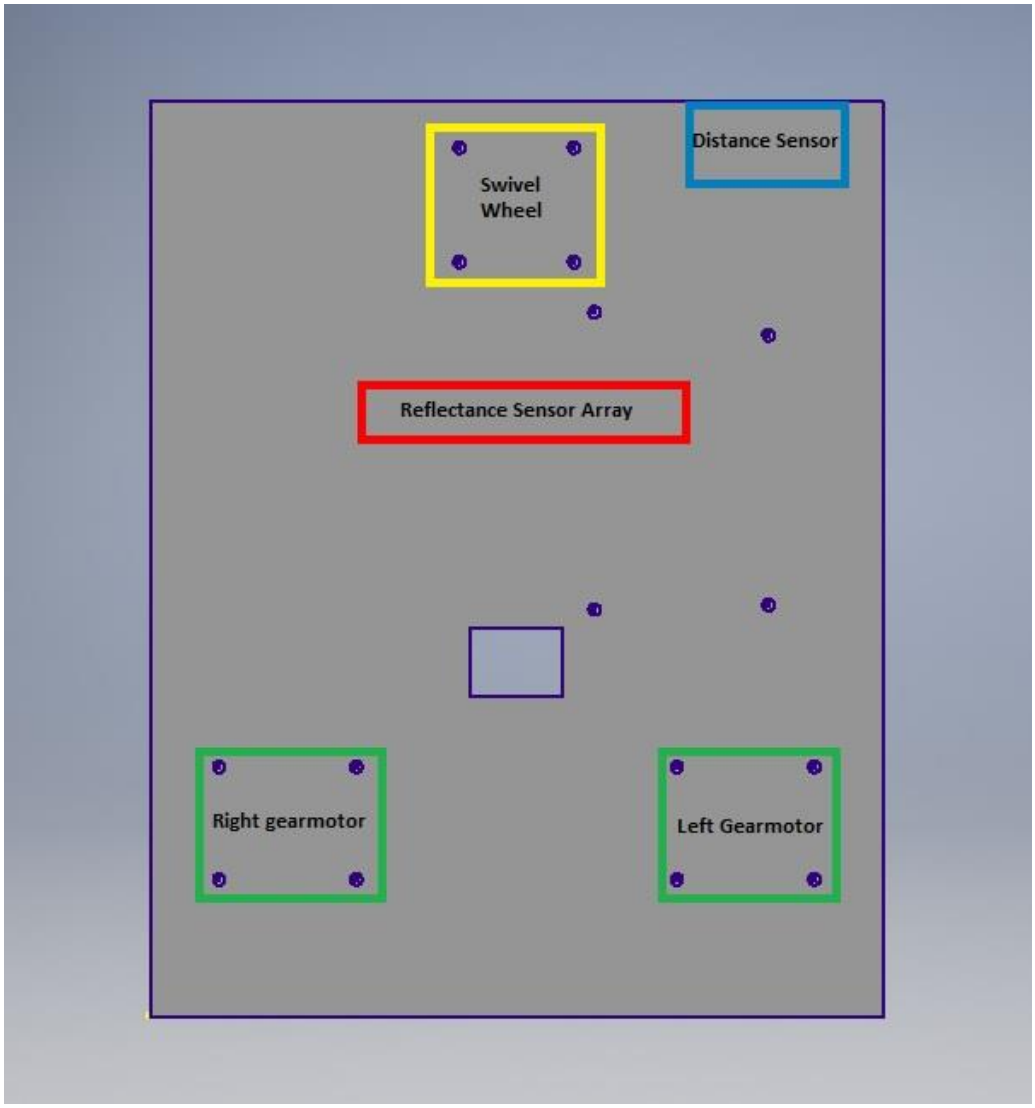

Figure 11. Bottom View of Chassis

On the top side of the chassis, the battery pack sits near the back of the robot. Moving towards the front of the robot is the squared opening that allows the wires coming from the gearmotors and the reflectance sensor array to be connected to the microcontroller. Nearby is the microcontroller, sitting on four screws with plastic spacers, and next to it is a mini breadboard attached using a sticky pad. Next to them is the linear actuator which was attached using hot-melt adhesive. Towards the front is the ring holder that prevented rings from falling to the sides once they were lifted. Figure 12 shows a pictorial representation of the top view of the chassis. Functionality of the two subsystems is explained below.

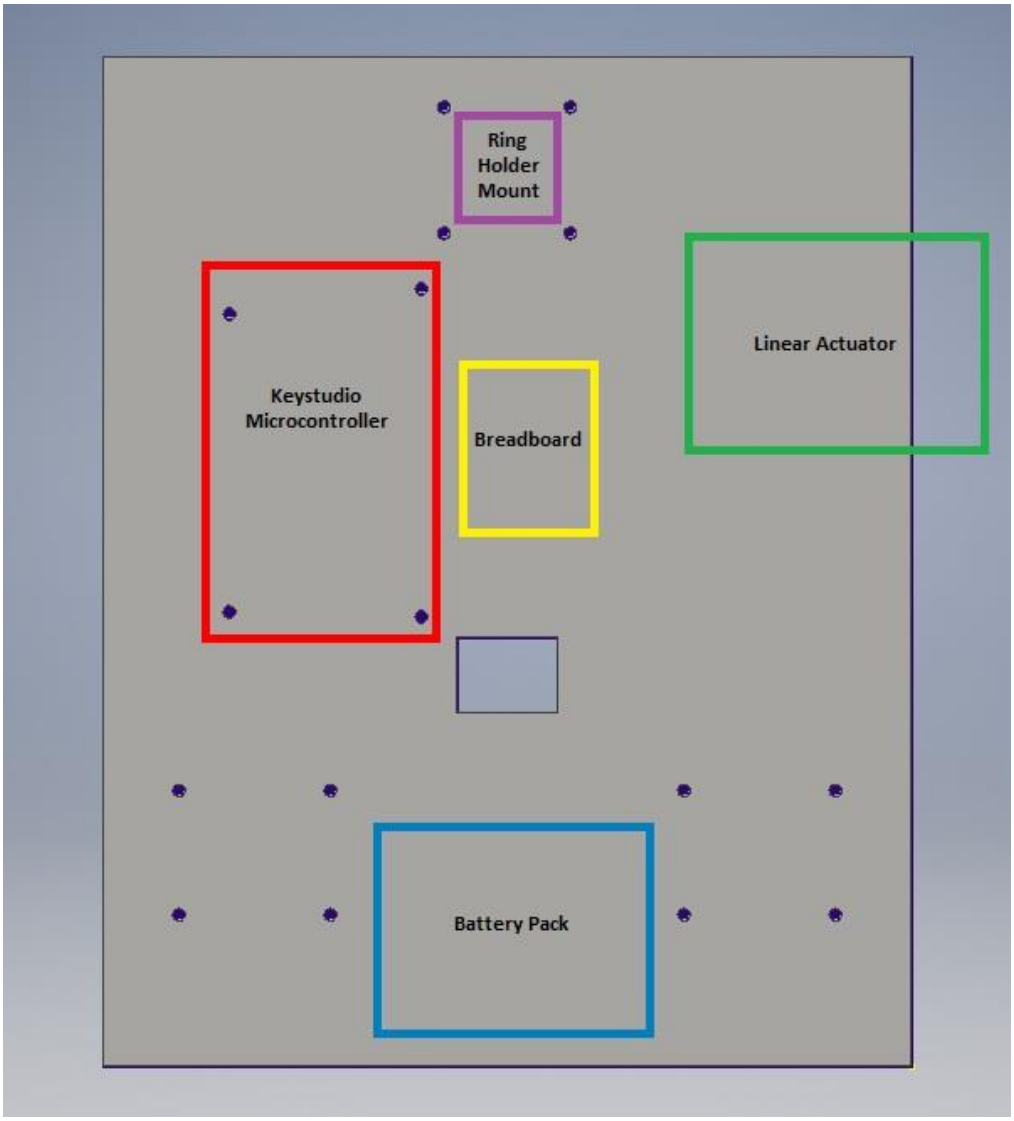

Figure 12. Top View of Chassis

Navigation Subsystem

The robot used a differential driving system with two independently controlled wheels on each side near the back of the chassis and a swivel wheel in the center front. The gearmotors were attached to the chassis using Pololu mounting brackets. Four #4-40 machine screws held each mounting brackets to the chassis and three M3 held each gearmotor to its mounting bracket. Attached to each gearmotor were Pololu mounting hubs which were screwed onto Pololu 90x10mm wheels. The front of the chassis was supported by a reused office chair swivel wheel. Most of these components can be seen in Figure 13 .

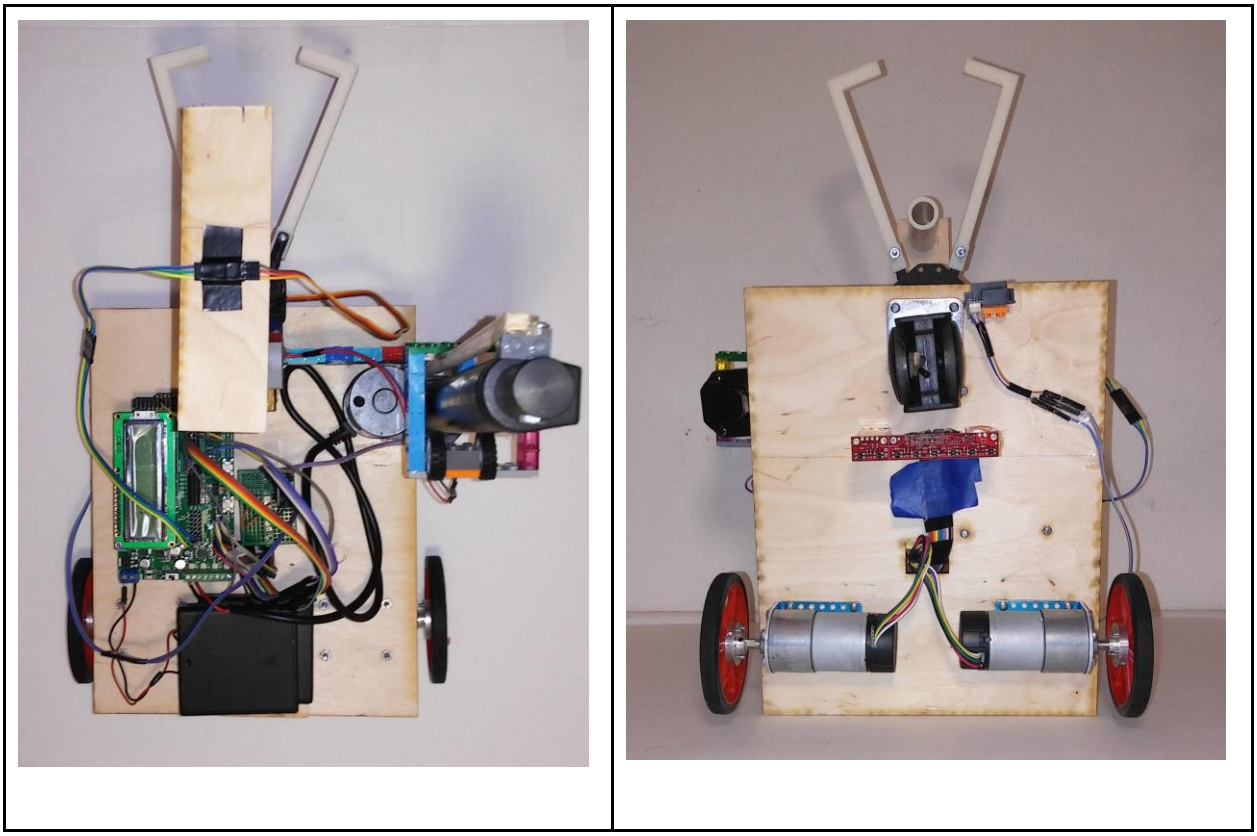

Figure 13. Top and Bottom Robot View.

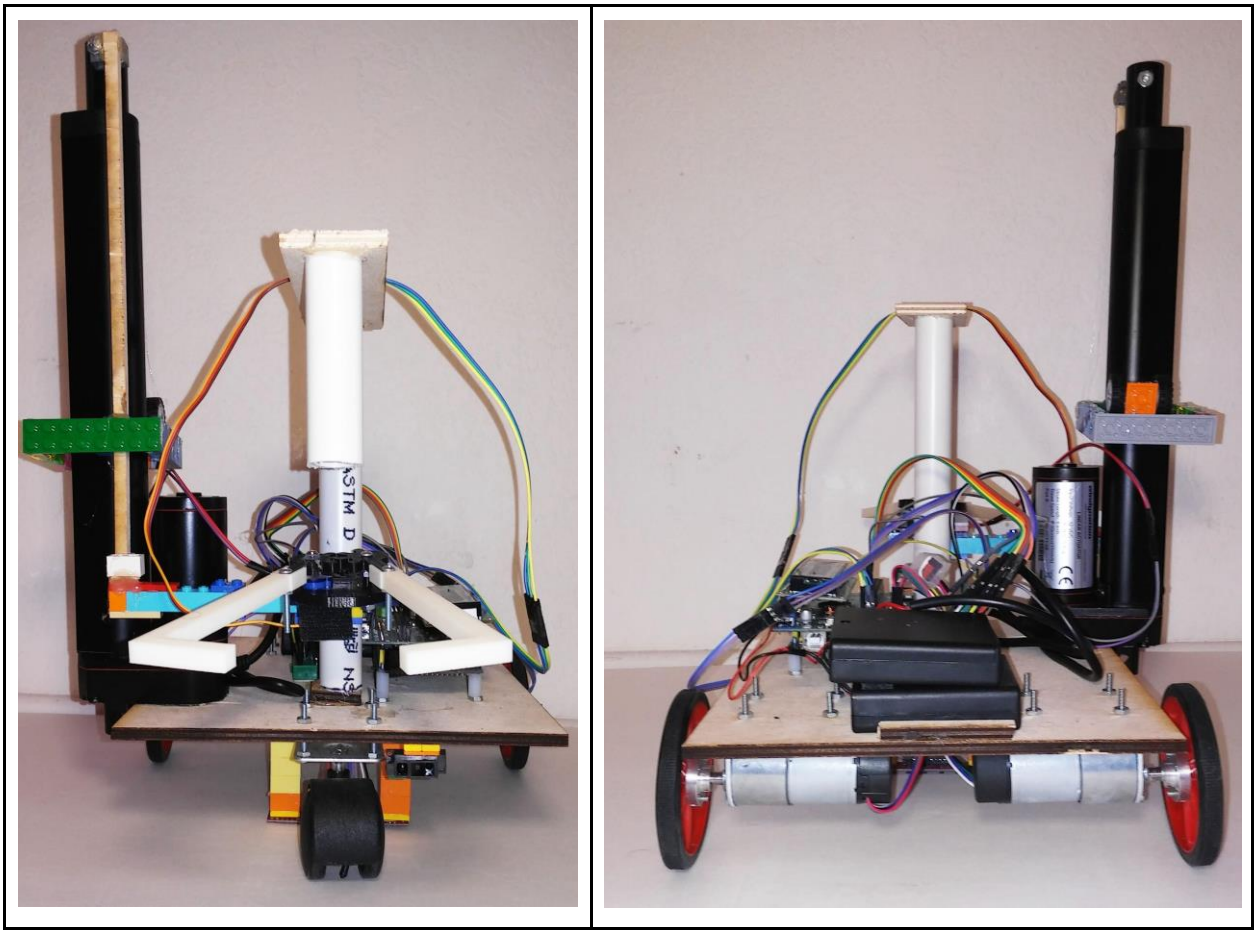

Figure 14. Front and Back Robot View.

Ring Pick-up/Drop-off

The most intricate mechanism of the robot is surely the ring-handling system. The initial approach was to build a pulley system using LEGO pieces. However, it was difficult to achieve structural integrity and consistency. Because the ring gripper needed to be supported from its backside, it would cause it to tilt forward because of the weight distribution. In other words, it was difficult to maintain the gripper in a horizontal position. The Windynation 8" Linear Actuator was used instead. The actuator raises and lowers a 3D-printed gripper. The original 3D-printable files were found and retrieved from Thingiverse.com [4]. Longer pinchers were designed to better grab rings. The gripper is actuated by the PowerPro SG90 9g Mini Servo and attached to the linear actuator using LEGO pieces and laser cut wood. As rings are raised by the gripper, they are positioned around a "ring holder", made by gluing two PVC pipes to a wood platform. The long support pipe is glued to the chassis and the small ring-holding pipe is meant to be positioned right above a scoring/supply peg. This prevents any of the top rings from falling over. The ring holder can be appreciated in the side views of the robot in Figure 15.

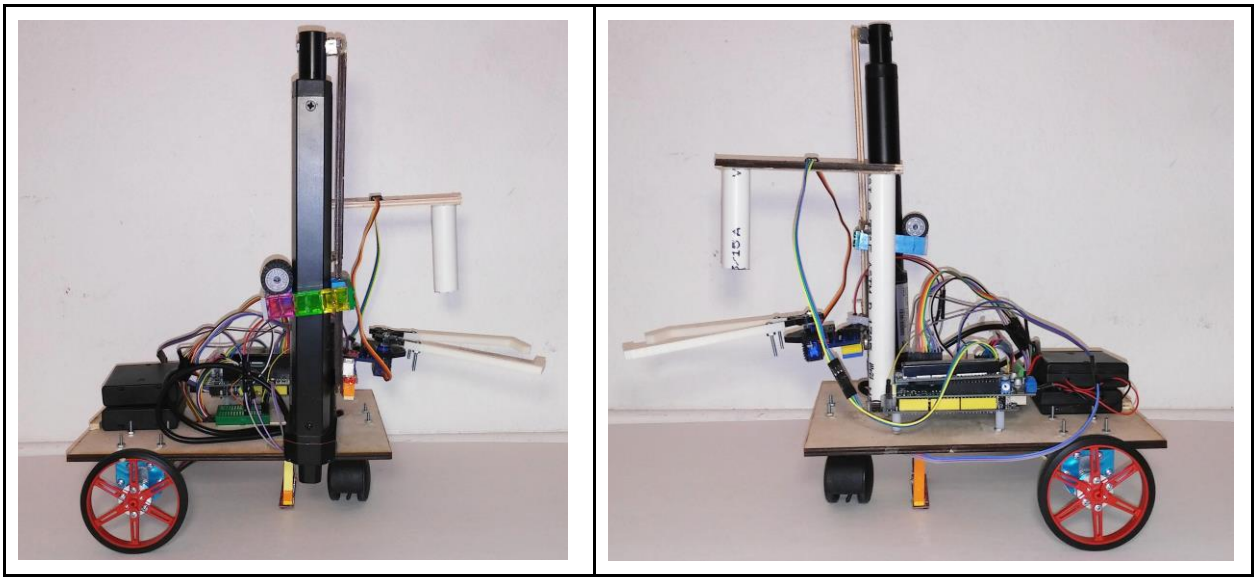

Figure 15. Left and Right Side Robot View.

Lastly, there is a reed switch attached to the gripper and there are two magnets hot-glued to the ring holder's support pipe; one is near the bottom signaling the reed switch that the gripper is low enough to grab onto rings, and the other is at a height at which the gripper clears the scoring/supply peg so that the robot can move away from it. **Error! Reference source not found.** shows a close-up photograph of this system.

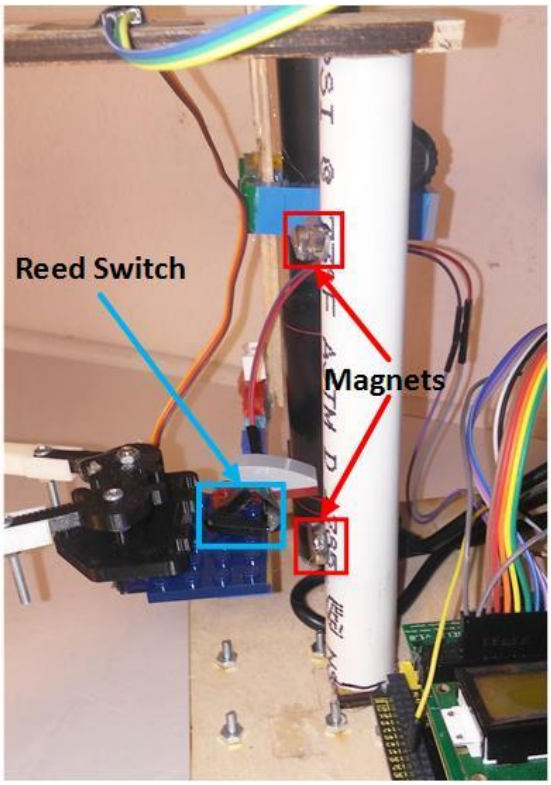

Figure 16. Reed Switch and Magnet Height Detection System

Budget and Bill of Materials

The Cal Poly Computer Engineering department funds up to $200 of senior project costs. Any more money spent past that is out-of-pocket for project members. As mentioned in the 'Mechanical Design' section, because of the relative high cost of linear actuators, it was initially attempted to take a less costly approach. However, given that it was not as reliable and mechanically stable as needed, it was decided to spend money past the original budget. It is also worth noting that the cost of the metal gearmotors was subsidized by Roborodentia, saving about $60 per pair. Below is the final bill of materials.

Table 1. Bill of Materials

Part Name	***Model***	***Supplier Name***	***Qty.***	***Unit Price ($)***	***Extended Price ($)***
Keystudio Microcontroller	Mega 2560 R3	Amazon.com	1	$14.99	$14.99
Battery Holder	4-AA Batteries, On-Off Switch	Amazon.com	1	$4.54	$4.54
Jumper Wire	40-pin Male to Male /Female to Female /Male to Female	Amazon.com	1	$8.99	$8.99
Linear Actuator	WINDYNATION 8" 12V 225lbs	Amazon.com	1	$59.99	$59.99
Batteries	Energizer Rechargeable - 8pk	Best Buy	1	$25.91	$25.91
Chassis Wood	1/4" x 12" x 12"	Cal Poly Bookstore	1	$3.17	$3.17
Metal Gearmotors	50:1 37Dx70L mm with 64 CPR Encoder	Cal Poly Robotics Club	2	$10.00	$20.00
Mounting Hubs	6mm Shaft, #4-40 Holes (2-pack)	Cal Poly Robotics Club	1	$7.00	$7.00
Mounting L-Bracket	Stamped Aluminum, for 37D mm Metal Gearmotors (2-pack)	Cal Poly Robotics Club	1	$8.00	$8.00
RoboShield	Version 1.0	Cal Poly Robotics Club	1	$35.00	$35.00
Reflectance Sensors	QTR-8A	Pololu.com	1	$9.95	$9.95

Analog Distance Sensor	Sharp GP2Y0A51SK0F	Pololu.com	1	$5.56	$5.56
Pololu Wheel	80mm x 10mm Pair - Red	Pololu.com	1	$9.25	$9.25
Reed Switch	RS-01C	Sparkfun.com	1	$1.95	$1.95
3D Printing & Laser Cutting	Digital Fabrication Lab Punch Card	Cal Poly CAED	1	$40.00	$40.00
Mini Servo	TowerPro SG90 9g	Amazon.com	1	$2.99	$2.99
Screws and Nuts	#4-40 and M3	Pololu.com	1	$4.00	$4.00
Total					**$261.29**

Lessons Learned

This project taught me many lessons along the way, most of them regarding robotics. Below are a few that are worth highlighting.

LiPo vs NiMH Batteries

Going into the project I had no knowledge of the different types of batteries and their specific behavior/performance. After reading the voltage requirement of around 10V for the RoboShield and the gearmotors, the first thing that came to my mind was to use 8 AA-batteries, and even better, to save a few dollars and the environment by choosing rechargeable ones. I soon learned that rechargeable batteries only provide 1.2V when fully charged. To top it all off, all the initial testing was done with a power supply so I did not get to experience or evaluate the performance of the rechargeable AA-batteries until the day before the competition. The issue with NiMH batteries is that under high-drain use, their output voltage drops at a high rate [5]. While most of the components do alright with lower a supply voltage, the gearmotors change the speed at which they turn for the same software-specified value. This meant that as the batteries were used more, the line following algorithm behaved differently, the linear actuator was slower, and the servo on the gripper had less torque. To drive the point home, every single returning team at the competition that I talked to was using a LiPo battery pack, and while I saw a handful of teams using AA-batteries, I believe I was the only ones using rechargeable ones.

Component price should be a considered factor but not a decisive one

During early design stages of the project, the idea of purchasing a linear actuator was scratched out because of their relatively high prices. In the end, it turned out that the time spent trying to build a substitute for a lower price was not worth the savings of not buying a linear actuator. So, when switching to the linear actuator, there were a few tweaks that had to be made in order to incorporate it. Most notably, there was no place to mount it through the chassis, which resulted in having it mounted on the side. At the same time, the gripper had to be awkwardly attached to the actuator, and weight distribution in the chassis was negatively affected; a firm push to the actuator would actually cause the robot to tip over. Overall, if the benefits of pricier components would have been weighted heavier than their price, system integration would have gone a lot smoother.

Theory does not equal practice

There were a couple of instances along the way in which mathematical models and datasheet information turned out to be notably different in practice. The biggest example was the distance-specific movement functions. Even though the motors spun the wheels for at least the required distance, there was extra time between the function

call to stop the motors and the time that they physically stopped. The wheels were often moving at different speeds or stopped at different times so one of the wheels would end up ahead of the other, resulting in a slight alteration of the robot's path. While the mathematical models can provide good approximations, there are always little deviations that need to be accounted for in practice.

Conclusion

In general, I believe the approach and the design of the robot were adequate; however, mechanical integration and power system came a little short. The ring pick-up system required the ring holder to be accurately positioned right above the scoring or supply peg. Even if the holder was misaligned from the supply peg by a few centimeters, there was a chance that the rings would hit the ring holder on their way up and be dropped by the gripper. Likewise, misalignment with a scoring peg often resulted in the rings not falling into the peg. This shortcoming combined with the fact that the battery voltage and, consequently, motor speeds were different between runs, resulted in unsuccessful ring pick-up/drop-off more often than desirable. A possible solution to this problem, used by a number of other competitors, would have been to move the robot forward until the front of the chassis is flush with the course wall so that the distance to the peg is always the same. Combining this with a LiPo battery pack to minimize motor speed variations between test runs would have increased the performance of the robot.

References

[1] Seng, John. “Roborodentia XXI”. Internet: https://docs.google.com/document/d/1rYSVdBPb6dNHQXfgRWYkEfzj00706zCozCtoAm_0bv4/pub, Jan. 11, 2016 [June 5, 2016]

[2] Gomberg, Brian. Seng, John. “roboshield”. Internet: https://github.com/roboshield/roboshield, Feb. 14, 2016 [June 8, 2016]

[3] Pololu. “Sharp GP2Y0A51SK0F Analog Distance Sensor Datasheet” Internet: https://www.pololu.com/file/0J845/GP2Y0A41SK0F.pdf.pdf, n.d. [June 6, 2016]

[4] Thingsverse. “Mini servo gripper”. Internet: http://www.thingiverse.com/thing:2415, Apr. 20, 2010 [June 5, 2016]

[5] Energizer Holdings. “Product Datasheet: Energizer NH15-2300” Internet: http://data.energizer. com/PDFs/nh15-2300.pdf, n.d. [June 5, 2016]

Appendix A: Roborodentia XXI Rules

Competition Rules and Course Specification:
Version 1.0 (1/11/16)
This year's competition is a head-to-head double elimination tournament where the object of the competition is to move rings onto scoring pegs.

Competition Date: April 16, 2016 (1pm)
Competition Location: Rec Center Gym, Cal Poly Campus

Teams are required to register with their intent to compete in Roborodentia. Registration forms will be made available.
Note: Rules are subject to minor updates and clarifications. Any changes will be announced and noted at the bottom of this page.
Download the competition layout here: Roborodentia 2016 Sketchup
1. Course Specifications (see attached diagrams for more details and dimensions)
1.1 The entire course is 8' wide x 8' long with 4" high walls surrounding the edges and along the center.
1.2 The black lines shown on the playing field are strips of 3/4" black masking tape.
1.3 There are 3 supply pegs (1/2" Sch 40 PVC pipe) located at the end of the field. Each supply peg will initially hold 4 **primary** (red or blue) rings.
1.4 At the left and right ends of the field are 3 scoring pegs (1/2" Sch 40 PVC pipe). Each peg is 3" tall.
1.5 There will be a box of **secondary** (green) rings. This box will contain 10 rings that will be randomly placed in the box.
1.6 Rings are painted PVC pipe (2" Sch 40 PVC pipe color red/blue/yellow/green and 1/2" pipe length).
2. Robot Specifications
2.1 Robots must be fully autonomous and self-contained.
2.2 Robots must have an 12" x 12" footprint or smaller at beginning of the match, but may autonomously expand after the match begins. At any point during a match, a robot's footprint may not be larger than 14" x 14".
2.3 A robot may have a maximum height of 15" at the start of a match. There is no height restriction after the match begins.
2.4 A robot may not disassemble into multiple parts.
2.5 Robots may not use any RF wireless receivers/transmitters during the competition.
2.6 Robots may not damage the course or the contest rings.
2.7 Adhesives may be used to pick up rings, but the rings may not be modified in any way. A ring must be completely free of residue after it has been picked up.
2.8 If a robot has RF wireless components on-board, the contestant will be required to notify the judges before the competition, and be able to demonstrate that the wireless components are not used. If RF components are found on-board that were not declared, or declared non-operational when active, it will be grounds for immediate disqualification.
2.9 Intentionally jamming an opponent's sensors is not allowed. Robots may not have weaponry or devices designed to damage or impede the operation of an opponent's robot.
2.10 A robot may not disturb rings on an opponent's side of the field.
2.11 A robot may not fly.
3. General Regulations
3.1 At the start of a match, a robot must be touching the tape intersection nearest to the supply pegs. The robot may start in any orientation.
3.2 Robots will be seeded based on qualifying runs.

3.3 The tournament will be run in a double elimination format.
3.4 A match will last 3 minutes.
3.5 If both teams agree, the match may end prior to three minutes.
3.6 At the end of a match, the robot with more points wins the match.
3.7 A team may pick up and restart their robot (touching the tape intersection nearest to the supply pegs) during the match. If a restart occurs, the opposing team will be awarded a bonus. (3 points for the first restart, 4 points for the second, and 5 points for the third, etc.)
3.8 On a restart, all rings will be removed from the robot.
3.9 A 3 second tone countdown will signal the start of a match. Contestants must start the robot during this period by pressing only 1 button 1 time. Contestants may not touch a robot during a match (except on a restart). Not restarting a robot ends the run for that robot and the robot keeps all points up to that instant.

4. Competition Regulations

4.1 Robots may start with 1 ring (primary or secondary) pre-loaded on the robot. Each supply peg will initially hold 4 rings.
4.2 If a contest ring goes off the playing field, then the ring is out of play with no penalty assessed.
4.3 A supply peg is replenished with up to 4 rings once a robot touches the center intersection.
4.4 The box of secondary rings will be replenished with up to 10 rings when there are 3 or fewer secondary rings in the box AND either:

- a robot scores a ring on scoring pegs #2
- a primary or center ring is scored

4.5 Rings that are dropped by a robot on its own side of the playing field will be removed by the judges when practical.
4.6 The first 2 rings that land on an opponent's side of the field will not be assessed a penalty. For each team, any rings after the first 2 will be assessed a 2 point penalty per ring (deducted from the robot corresponding to the ring color). Opponent rings resting on a team's playing field will be removed by the judges as soon as practical.

5. Scoring

5.1 Primary rings (red or blue) that are placed on scoring pegs #1 will be worth 1 point each.
5.2 Primary rings that are placed on scoring pegs #2 will be worth 3 points each.
5.3 Center rings (yellow) on their own are worth 3 points. When a center ring is part of a stack of rings on a scoring peg, the point value of the stack will be tripled (a center ring will count towards the score as if it were a primary ring). This triple bonus is applied only once per stack even if there are multiple center rings in a stack.
5.4 Secondary rings (green) will be worth 2 points on scoring pegs #1. Secondary rings will be worth 8 points on scoring pegs #2.
5.5 Once a scoring peg holds 4 or more rings, the rings will be removed as soon as practical. Only the first 4 rings on a peg will count towards the score (any rings above the 4th ring will not count).
5.6 At the end of the match, any primary rings located in the 2nd position of a center peg will be worth 20 points.
5.7 Every pair of primary rings placed in the secondary ring box is worth 1 point.

6. Penalties

6.1 If any part of a robot breaks the plane of the center wall that is farthest from the robot, then that robot will be assessed a penalty. This allows a robot to be directly over the center wall without penalty. Otherwise, the penalty is that all rings for that robot on the center peg are invalidated, the robot's scored is multiplied by .5, and the match ends for that robot.
6.2 A robot that attempts to damage an opponent's robot will be disqualified for that match.
6.3 Robots that do not move within the first 20 seconds of a match will be considered inoperable and will forfeit the match.

6.4 If both robots have not moved for 60 seconds (at any time during a match), the match will end.
6.5 If a robot exceeds the size restrictions during a match, the match ends for that robot. The opponent robot may continue the match.

7. Tie breakers

In the event of a tie, the following tie breakers (listed in order below) will be used to determine a winner:

1. Whichever robot scores more center rings
2. Whichever robot scores more secondary rings
3. Whichever robot removes more center rings from pegs
4. Whichever robot removes more primary rings from pegs
5. One round of rock, paper, scissors
6. Coin toss

8. Contestant eligibility

Current university students and former students that graduated during the 2015-2016 academic year may enter Roborodentia XXI.

9. Prizes

Below are the prize levels:
1st Place - $1,000
2nd Place - $600
3rd Place - $400

Appendix B: RoboShieldMain.ino

```
#include <RoboShield.h>
#include <RoboShield_Defines.h>
#include <Wire.h>

//PINS
#define GRIPPER_PIN 0
#define REED_SENSOR_PIN 53
#define LEFT_MOTOR_PIN 2
#define RIGHT_MOTOR_PIN 3
#define IR_SENSOR_PIN 8
#define IR_SENSOR1_PIN 1
#define ACTUATOR_PIN 0
#define RIGHT_ENCODER 1
#define LEFT_ENCODER 0
#define OUTER_RIGHT 0
#define CENTER_RIGHT 1
#define CENTER_LEFT 2
#define OUTER_LEFT 3

//Line following
#define BASE_SPEED 12
#define CENTER_CORRECTION 0.016     // 8/500
#define OUTER_CORRECTION 0.024      // 12/500
#define SLOW_MOTOR_CORRECTION -5

//Argument values
#define MOTOR_DISTANCE_SIX_IN 429
#define IR_DISTANCE_ONE_IN 390
#define ENCODER_TURN 1150
#define BLACK_TAPE_READING 450
#define GRIPPER_OPENED -45
#define GRIPPER_CLOSED 40
#define ACTUATOR_UP -100
#define ACTUATOR_DOWN 100

//Globals
RoboShield shield;
QTRSensor reading;
int QTRReadings[6];

//Prototypes
int FollowLine();
void TurnRight();
void TurnLeft();
void TurnAround();
void AdvanceUntilIR(int value);
void BackUpFor(int value);
void ReadQTR(QTRSensor *reading);
```

```
typedef struct {
  int outerRight;
  int centerRight;
  int centerLeft;
  int outerLeft;
} QTRSensor;


void setup() {
  //Set motor pins as outputs
  pinMode(LEFT_MOTOR_PIN, OUTPUT);
  pinMode(RIGHT_MOTOR_PIN, OUTPUT);
  pinMode(REED_SENSOR_PIN, INPUT);

  //Stop motors
  shield.setMotor(LEFT_MOTOR_PIN, 0);
  shield.setMotor(RIGHT_MOTOR_PIN, 0);

  //Open gripper
  shield.setServo(GRIPPER_PIN, GRIPPER_OPENED);

  //Wait for button press
  while (!shield.buttonPressed())
    ;
}

void loop() {

  LiftRing();

  BackUpFor(MOTOR_DISTANCE_SIX_IN);

  TurnAround();

  while(FollowLine())
    ;

  TurnRight();

  while (FollowLine())
    ;

  AdvanceUntilIR(IR_DISTANCE_ONE_IN);

  LowerRing();

  BackUpFor(MOTOR_DISTANCE_SIX_IN);

  TurnAround();

  while(FollowLine())
    ;
```

```
  TurnLeft();

  while (FollowLine())
    ;

  AdvanceUntilIR(IR_DISTANCE_ONE_IN);

}

int getMax(int num, int arr[]) {
  int i, max = 0;
  for (i = 1; i < num; i++) {
    if (arr[i] > arr[max])
      max = i;
  }

  return max;
}

void LiftRing () {
  shield.setServo(GRIPPER_PIN, GRIPPER_CLOSED);   //Close gripper/grab rings
  shield.setMotor(ACTUATOR_PIN, ACTUATOR_UP);     //Begin moving the linear actuator up
  delay(5000);                                    //Wait for the actuator to clear the bottom magnet
  while (digitalRead(REED_SENSOR_PIN) == 0)       //Wait for the reed switch to see the top
magnet
    ;
  shield.setMotor(ACTUATOR_PIN, 0);               //Stop the actuator
}

void LowerRing() {
  shield.setMotor(ACTUATOR_PIN, ACTUATOR_DOWN);   //Move actuator down
  delay(5000);                                    //Wait for actuator to clear the top magnet
  while (digitalRead(REED_SENSOR_PIN) == 0)       //Wait for the actuator to reach the bottom
    ;
  shield.setMotor(ACTUATOR_PIN, 0);               //Stop the actuator
  shield.setServo(GRIPPER_PIN, GRIPPER_OPENED);   //Open the gripper/drop rings
}

void BackUpFor(int value) {
  shield.resetEncoder(RIGHT_ENCODER);             //Reset encoder count
  shield.setMotor(LEFT_MOTOR_PIN, -BASE_SPEED);
  shield.setMotor(RIGHT_MOTOR_PIN, BASE_SPEED);     //Set motors to move backwards
  while(shield.readEncoder(RIGHT_ENCODER) < value)  //Wait for encoder count to reach
target
    ;
  shield.setMotor(RIGHT_MOTOR_PIN, 0);
  shield.setMotor(LEFT_MOTOR_PIN, 0);             //Stop the motors
}

int FollowLine() {
  static double rightCorrection, leftCorrection;
  static int max;
```

```
//Read reflectance sensors
ReadQTR(&reading);

//Reset corrections
leftCorrection = 0;
rightCorrection = 0;

//Lock for horizontal line
if (reading.midRight > BLACK_TAPE_READING &&
    reading.midLeft > BLACK_TAPE_READING &&
    reading.centerLeft > BLACK_TAPE_READING &&
    reading.centerRight > BLACK_TAPE_READING) {

  shield.setMotor(LEFT_MOTOR_PIN, 0);
  shield.setMotor(RIGHT_MOTOR_PIN, 0);
  delay(100);
  return 0;
}

//Check if going straight
else if (!(reading.centerLeft > BLACK_TAPE_READING &&
         reading.centerRight > BLACK_TAPE_READING)) {

  //Find minimum
  max = getMax(4, QTRReadings);

  //Find the sensor closest to the black tape line
  switch(max) {
    case OUTER_RIGHT:
      //Speed up opposite motor
      leftCorrection = OUTER_CORRECTION * QTRReadings[max];
      //Slow down the same-sided motor
      rightCorrection = SLOW_MOTOR_CORRECTION;
      shield.lcdClear();
      shield.lcdPrintf("ORight");
      break;

    case CENTER_RIGHT:
      leftCorrection = CENTER_CORRECTION * QTRReadings[max];
      rightCorrection = SLOW_MOTOR_CORRECTION;
      shield.lcdClear();
      shield.lcdPrintf("CRight");
      break;

    case CENTER_LEFT:
      rightCorrection = CENTER_CORRECTION * QTRReadings[max];
      leftCorrection = SLOW_MOTOR_CORRECTION;
      shield.lcdClear();
      shield.lcdPrintf("CLeft");
      break;

    case OUTER_LEFT:
      rightCorrection = OUTER_CORRECTION * QTRReadings[max];
```

```
          leftCorrection = SLOW_MOTOR_CORRECTION;
          shield.lcdClear();
          shield.lcdPrintf("OLeft");
          break;

        default:
          shield.lcdClear();
          break;
      }

      //Update motor speeds
      shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED + (int)leftCorrection);
      shield.setMotor(RIGHT_MOTOR_PIN, -BASE_SPEED - (int)rightCorrection);
    }

    //Keep going straight
    else {
      //Set both motor speeds equal
      shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED);
      shield.setMotor(RIGHT_MOTOR_PIN, -BASE_SPEED);
      shield.lcdClear();
      shield.lcdPrintf("Straight");
    }

    return 1;
}

void TurnAround() {
    //Reset encoder
    shield.resetEncoder(RIGHT_ENCODER);

    //Begin turning
    shield.setMotor(RIGHT_MOTOR_PIN, BASE_SPEED);
    shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED);

    //Turn for little less than 180deg
    while (shield.readEncoder(RIGHT_ENCODER) < 400)
      ;

    //Find the black tape line
    while(shield.getAnalog(4) < 300)
      ;

    //Stop
    shield.setMotor(RIGHT_MOTOR_PIN, 0);
    shield.setMotor(LEFT_MOTOR_PIN, 0);
}

void AdvanceUntilIR(int value) {
    //Set motors to move forward
    shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED);
    shield.setMotor(RIGHT_MOTOR_PIN, -BASE_SPEED);
```

```
  //Wait until specific IR reading
  while (shield.getAnalog(IR_SENSOR_PIN) < value)
    ;

  //Stop motors
  shield.setMotor(LEFT_MOTOR_PIN, 0);
  shield.setMotor(RIGHT_MOTOR_PIN, 0);
}

void TurnRight() {
  shield.resetEncoder(LEFT_ENCODER);
  shield.setMotor(LEFT_MOTOR_PIN, BASE_SPEED);
  while (shield.readEncoder(LEFT_ENCODER) < ENCODER_TURN)
    ;
  shield.setMotor(LEFT_MOTOR_PIN, 0);
}

void TurnLeft() {
  shield.resetEncoder(RIGHT_ENCODER);
  shield.setMotor(RIGHT_MOTOR_PIN, BASE_SPEED);
  while (shield.readEncoder(RIGHT_ENCODER) < ENCODER_TURN)
    ;
  shield.setMotor(RIGHT_MOTOR_PIN, 0);
}

void ReadQTR(QTRSensor *reading) {
  QTRReadings[0] = reading->outerRight = shield.getAnalog(2);
  QTRReadings[1] = reading->midRight = shield.getAnalog(3);
  QTRReadings[2] = reading->centerRight = shield.getAnalog(4);
  QTRReadings[3] = reading->centerLeft = shield.getAnalog(5);
}
```