PI-x (Roborodentia)

Kelly Leung, William Luo, and Jeffrey Tang
Advisor: Bridget Benson
California Polytechnic State University, San Luis Obispo
Computer Engineering Program
June 13th 2015

ABSTRACT

Pl-x is the name of the robot that will be competing in Roborodentia, an annual autonomous robot competition. This year the competition involves moving rings from one end of a course to the opposite end of the course.

ACKNOWLEDGEMENTS

We would like to thank our advisor (Bridget Benson) and Jeffrey Gerfen for providing us with the knowledge that we needed about microprocessors and micro-controllers. They had given us a solid foundation and taught us what steps need to be taken in creating a project of our own.

A special thanks goes to John Seng. Not only does he commit a tremendous amount of his time to ensure that the competition goes as smooth as possible, but he has also provided us with the guidance we need in purchasing the necessary material. We greatly appreciate his dedication to his students.

We would also like to thank Hugh Smith - the professor that had introduced us to the Arduino and servos in CPE200, the class in which the three of us created a robotic teddy bear. Through this exposure we were familiar with the material that we needed in completing PI-x.

In addition, we would like to show our thanks to the Computer Engineer Society (CPEs) for opening up their room for us and allowing us to use their equipment.

Finally, we would like to thank the industry sponsors for making this competition possible every year and the Computer Engineering department for providing us with the funding that we needed for our robot.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

INTRODUCTION

Pl-x, the robot, will be competing in the Annual Robot Competition hosted by Eta Kappa Nu (HKN), Cal Poly's Electrical and Computer Engineering Honor's Society. The competition will be held in Spring 2015 (April 18th) in the Mott Gym at Cal Poly's Campus.

This year's competition will involve two robots, each on their own side of the course (which can be seen in Figure 1), moving rings across the field and placing them onto pegs. The rings will be moved from the horizontal pegs to the vertical pegs. Each match will last three minutes. The robot that has the most points is the winner of that round. Please refer to the Requirements (Specification to Win) section to see how the points are obtained.

The first, second, and third place winners of the competition will win $1000, $600, and $400 respectively.

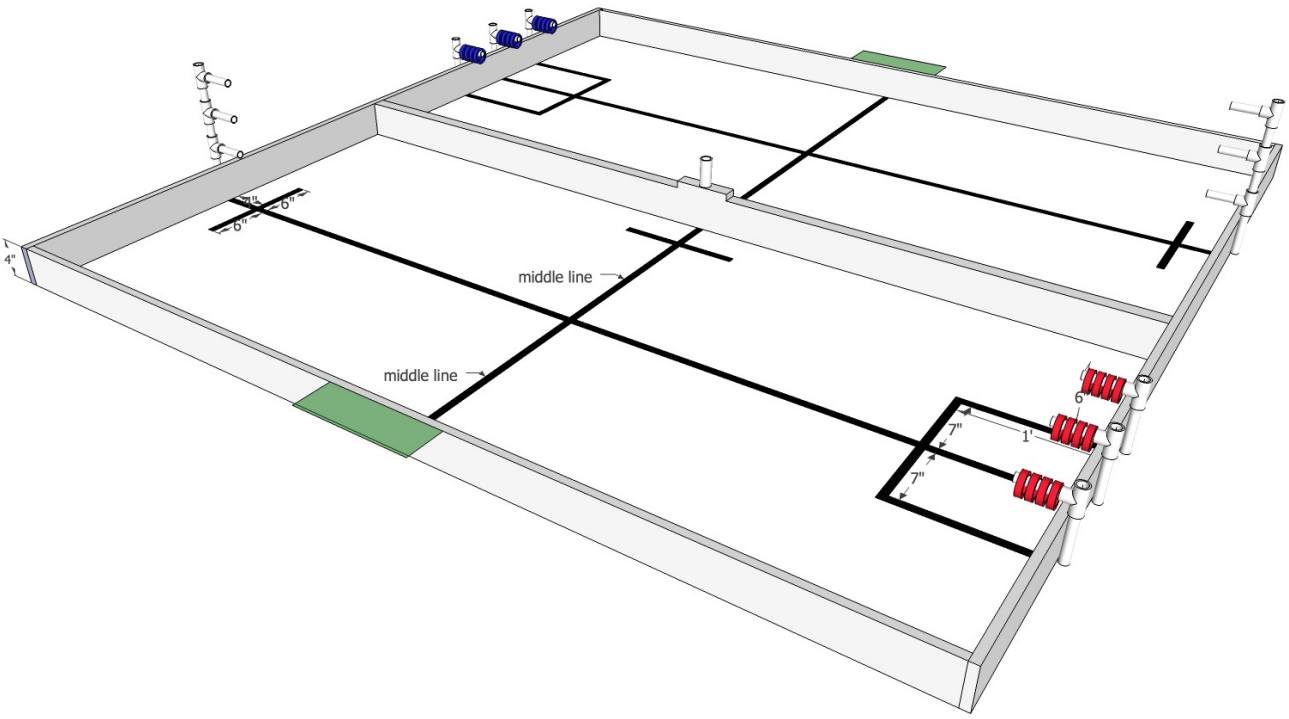

Figure 1: Roborodentia Course

REQUIREMENTS

In order for a fair competition, there are several specifications that all robots must follow. These specifications were ones that we had taken to account when construction our robot. The specifications are as follows.

Robot Specifications

1. Fully autonomous
2. Footprint is at most 11” x 11” at the start of the match; may expect after the match has begin, but can be no larger than 13” x 13”
3. Maximum height of 12” at the start of the match; there is no height restriction after the match begins
4. Cannot disassemble into multiple parts
5. No RF wireless receivers/transmitters can be used
6. Cannot damage the course or the contest rings
7. Adhesives cannot be used to pick up the rings, but no residue can be left behind
8. If RF wireless components are used, the contestant must notify the judges before the start of the competition and be able to demonstrate that the components will not be used
9. No weapons are allows nor is intentionally jamming the opponent’s sensors
10. Rings on the opponent’s side may not be disturbed

Specifications to Win

Obtain the most amount of points against your opponent. Points are earned based on the following rules:

- Rings are scored once they stay on a scoring peg for 3 seconds without support from a robot
- Rings placed on the lowest, middle, and highest scoring pegs are worth 2, 4, and 6 points respectively
- When a robot has rings on all 3 scoring pegs simultaneously, it will be awarded a 14 point bonus once the robot returns to touch the middle line; at that time, all rings will be removed from the 3 scoring pegs
- The robot with the highest ring that is completely seated on the center peg at the end of the match will have its overall score multiplied by 2
- A ring dropped into the scoring zone outside the field (shown in green) is worth 1 point; a maximum of 20 points may be scored using this scoring zone

DESIGN

This section describes the hardware and software design elements of PI-x. The hardware and software work together to transport rings from the horizontal supply pegs to the vertical scoring pegs. The act of transporting the rings is broken down into three main parts: picking up the rings, following the line across the course, and safety unloading the rings.

System Architecture

PI-x consist of 2 switches, a sensor array, an Arduino Mega 2560, 1 motor driver, 3 servos, and 2 motors, as can be seen below in Figure 2. The 2 switches are used for wall detection; they are mounted on their respective sides and will trigger upon impact against the course wall. The reflectance sensor array has 8 IR LED/phototransistor pairs which are used for line detection. The Arduino Mega interprets the switches and sensors to decide the positioning and speed of the servos and motors. The motor driver (which drives the two motors) and the Arduino are powered by their connections to a 12-volt battery. The Arduino then outputs 5-volts to the servos and sensors.

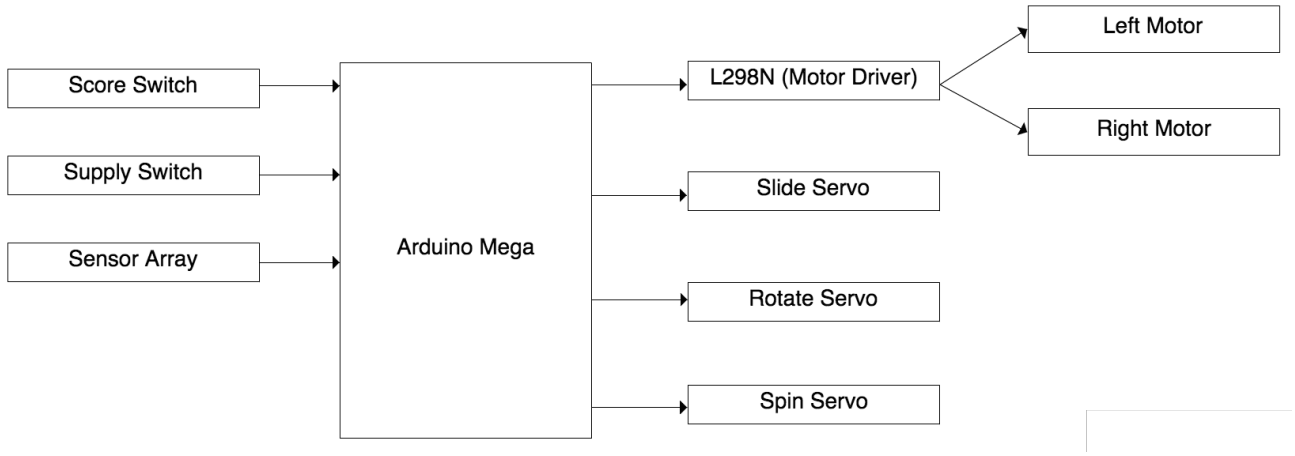

Figure 2: High Level Black Box Diagram

Table 1 shows what tasks are handled by the hardware or the software.

Table 1: Partitioning of Hardware and Software Tasks

Hardware	Software
Provides power to the components	--
Sets input value with switches	Interprets switches input values and reacts accordingly
Reflectance sensor senses line deviation	Interpret digital values and speeds up motors accordingly
Motor driver allows motors to be driven bidirectionally	--

Hardware

HARDWARE COMPONENTS

- Arduno Mega 2560
- Roller Lever Switch (2)
- Pololu QTR-8RC Reflectance Sensor Array
- L298N H-Bridge Motor Driver
- Hitec 35645S HS-5645MG Digital Hi Torque Metal Gear BB Servo (2)
- Hitec 33322S HS-322HD Standard Deluxe Karbonite Gear Servo (2)
- 70:1 Metal Gearmotor 37Dx54L mm (2)
- Pololu Ball Caster with 3/4" Metal Ball
- Pololu Wheel 80x10mm Pair - White
- Pololu Stamped Aluminum L-Bracket Pair for 37D mm Metal Gearmotors
- Pololu Universal Aluminum Mounting Hub for 6mm Shaft, #4-40 Holes (2-Pack)
- 8-AA Battery Holder
- PVC Pipes and Plexiglass
- Assortment of Nuts, Screws, Washers, Wires, Jumpers, and Resistors

Table 2 shows the pin connections between the 8 sensors and the Arduino. The ground of the sensors in connected to the Arduino ground. The Vcc of the sensors is connected to the 5-volt output of the Arduino.

Table 2: Sensors to Arduino Connection

Sensor	Pin Name	Arduino Pin	Port Name
Sensor 0	PIN 1	DIGITAL IN: 31	PC6
Sensor 1	PIN 2	DIGITAL IN: 33	PC4
Sensor 2	PIN 3	DIGITAL IN: 35	PC2
Sensor 3	PIN 4	DIGITAL IN: 37	PC0
Sensor 4	PIN 5	DIGITAL IN: 39	PG2
Sensor 5	PIN 6	DIGITAL IN: 41	PG0
Sensor 6	PIN 7	DIGITAL IN: 43	PL6
Sensor 7	PIN 8	DIGITAL IN: 45	PL4

Table 3 shows the connections between the Arduino and the Motor Driver. The Motor Driver is used to control the speed and the direction of the left and right motor.

Table 3: Arduino to Motor Driver Connection

Motor Driver Pin Name	Arduino Pin	Port Name
motor1_enable	DIGITAL IN: 8	PH5
motor2_enable	DIGITAL IN: 9	PH6
motor1_cw	DIGITAL IN: 4	PG5
motor1_ccw	DIGITAL IN: 5	PE3
motor2_cw	DIGITAL IN: 6	PH3
motor2_ccw	DIGITAL IN: 7	PH4
GND	POWER: GND	GND

Table 4 shows the connections between the switches and the Arduino. PIN 1 is connected to a pull down resistor to ground of 10KΩ and PIN 2 is connected to 5V out of the Arduino.

Table 4: Switches to Arduino Connection

Switch	Pin Name	Arduino Pin	Port Name
Switch 1	PIN 1	DIGITAL IN: 25	PA3
Switch 2	PIN 1	DIGITAL IN: 50	PB3

HARDWARE CONSIDERATIONS

Choosing some of the hardware components took careful consideration. The subsections below describe why we selected the hardware components we did.

Arduino Mega 2560

We decided to go with the Arduino platform due to familiarity and convenience of the available libraries. The Arduino Mega was chosen over the Arduino Uno due to the additional pulse width modulation (PWM) signals and digital inputs.

70:1 Metal Gearmotor 37Dx54L mm, Wheels, and Motor Driver

We realized the motor driver was needed to allow the motors to go both ways at varying speeds and independent power supplies form the signal wires. The 80 mm diameter wheels were chosen because of the increased tire width, which therefore increases traction. 80 mm will also provide us the ability to mount our motor on top of our base, providing the adequate amount of clearance for our sensor.

Because we wanted the most amount of torque, the first thing we looked for in a motor that would go at least 2 feet per second. Given that our wheels are 80mm in diameter, we needed 145.5 rpm motor.

The torque was calculated using *force x radius*. The force in this equation is the friction force between the wheels and the ground. The friction force is equal to *normal force x friction coefficient*. To simplify our calculations we assumed a friction coefficient of 1; this results in friction force being equal to normal force. The motor, which provided a 150 rpm and 14 kg-cm allows us to have a normal force of 15.87 lbs.

Pololu QTR-8RC Reflectance Sensor Array

This sensor array was very convenient due to its form factor and simplicity. Initially we thought about building our own circuit but decided that the sensor array meets the same needs and could be easily mounted.

Software

We wrote all the software for this project in Sketch. This section describes the software flow.

SOFTWARE FLOW

Pl-x is designed to pick up rings from one side of the course and deliver them to the other side. To do this in the most efficient way, Pl-x has 3 prongs. The center prong is spaced 4" and 7" from the middle prong. This allows Pl-x to either pick up or drop off all 3 sets out rings in two of their respective sequences. The fork is initially stored in a vertical orientation to not exceed the 11" footprint requirement.

After the match starts, the footprint of the robot can now be 13" and there is no height restriction. Pl-x raises its "Slide Servo" 1.5 inches up to clear the base height and freely rotate, as mentioned in Figure 3, the Software Flow Diagram. The fork is then rotated from its vertical orientation 90 degrees to a horizontal orientation, by rotating the "Rotate Servo". The horizontal orientation will align two of the prongs to their corresponding supply pegs. The servo pick-up sequence will pick up the rings from the two said supply pegs. After picking up the rings, Pl-x will back up 5 inches and slide its "Slide Servo" 3 inches to the right. Pl-x will go forward 5 inches and perform the pick-up sequence again to pick up the rings on the remaining supply peg.

At this time, Pl-x now holds all the supply rings from the supply pegs. Pl-x will spin its "Spin Servo" 180 degrees to face the scoring pegs. The "Rotate Servo" will also rotate back to its vertical orientation to match the scoring pegs. The robot will makes its way to the other side of the course, using the reflectance sensor as a line follower. Pl-x stops when the switch is triggered, indicating impact will the opposite side of the course.

The drop off sequence is then performed. Pl-x backs up 5 inches, slides the "Slide Servo" 3 inches to match with the remaining pegs and then moves back in to perform the drop off sequence again. The robot then spins 180 degrees and rotates 90 degrees to match the supply pegs and makes its way back to the supply rings to pick up more rings.

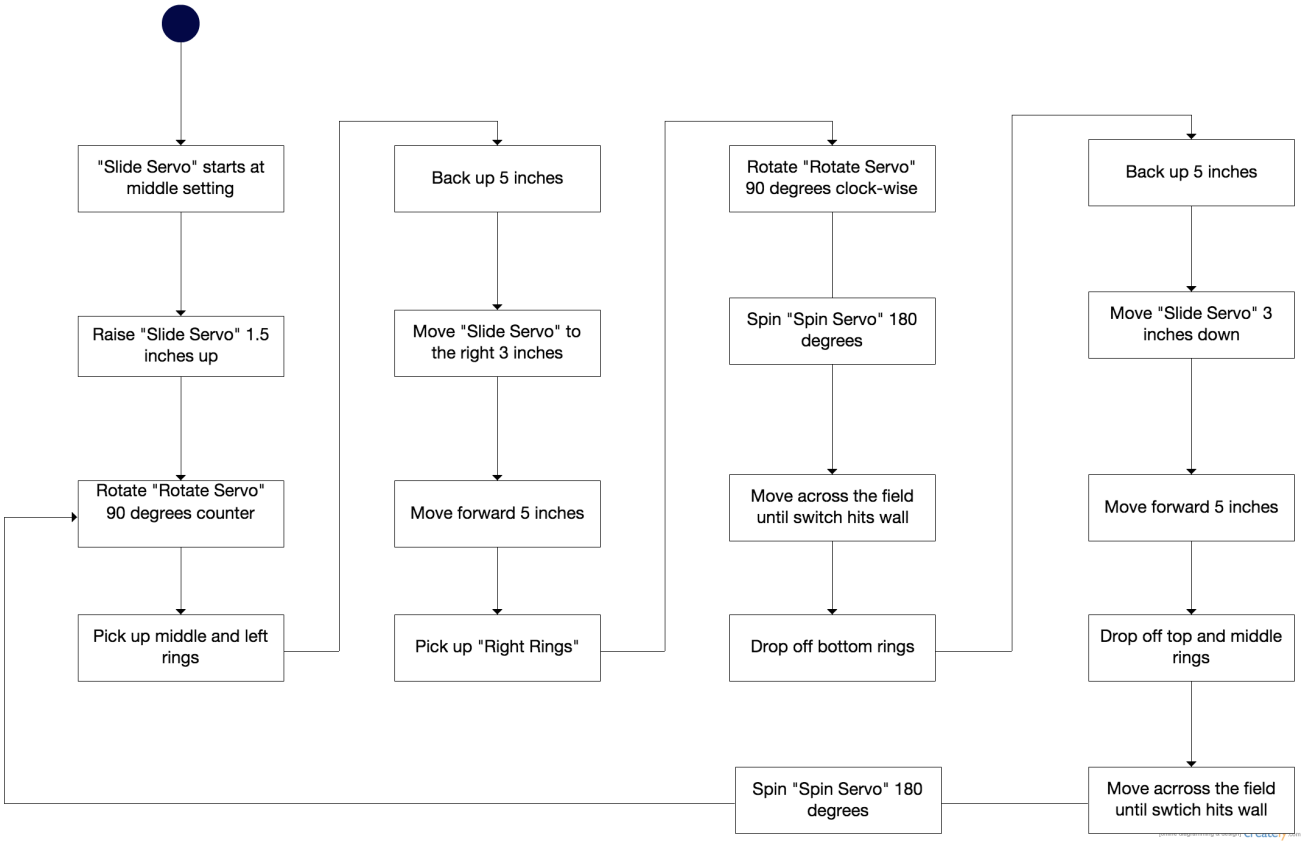

Figure 3: Software Flow Diagram

System Integration

Figure 4 shows the system integration of our design. Refer to tables 2-4 for the exact pin listings.

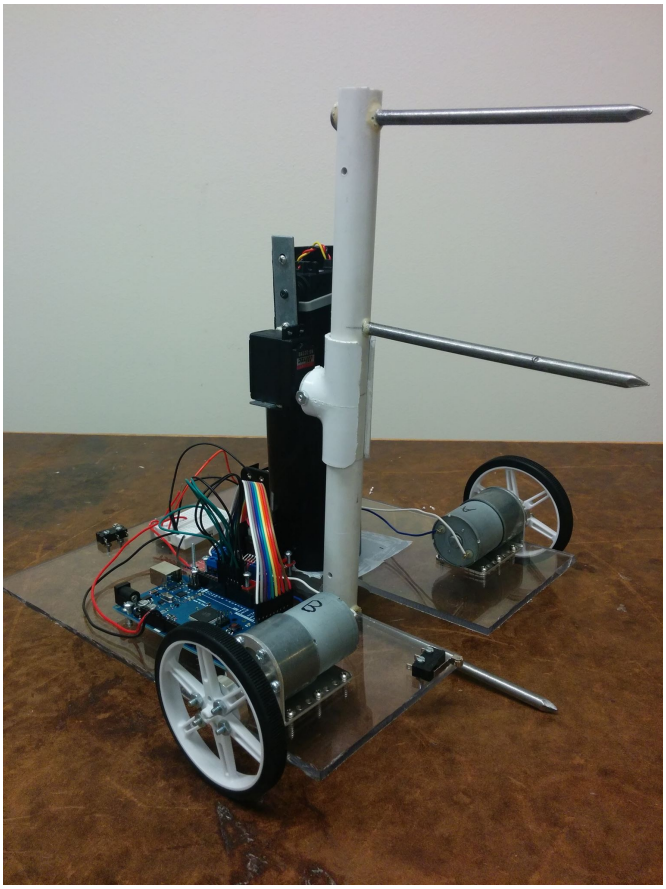

Figure 4: System Integration

Meeting Design Requirements

Table 5 below lists our design requirements and whether or not they were met.

Table 5: Design Requirements

Requirement	Requirement Met
Fully autonomous	Yes
Footprint is at most 11” x 11” at the start of the match; may expect after the match has begin, but can be no larger than 13” x 13”	Yes; 12” fork starts in vertical position to not exceed footprint limitations
Maximum height of 12” at the start of the match; there is no height restriction after the match begins	Yes; vertical form moves up after the match starts
Cannot disassemble into multiple parts	Yes
No RF wireless receivers/transmitters can be used	Yes; no RF components
Cannot damage the course or the contest rings	Yes
Adhesives cannot be used to pick up the rings, but no residue can be left behind	Yes; no adhesives are used
If RF wireless components are used, the contestant must notify the judges before the start of the competition and be able to demonstrate that the components will not be used	Yes; no RF components
No weapons are allows nor is intentionally jamming the opponent’s sensors	Yes; no weapons or jamming intended
Rings on the opponent’s side may not be disturbed	Yes
Winning	N/A; we have not yet competed in the competition

TESTING AND IMPROVISIONS

Testing the Hardware

In order to test the correct operation of our design, we performed the following tests shown in Table 6.

Table 6: Hardware Test Table

Test Tried	What We Expected	Results
Mounted a motor onto the base	The robot to move forward	Unsuccessful. There were no movements; the motor did not have enough torque to drive the weight of the robot
Mounted a new motor, which has 15x the amount of torque as the previous one	The robot to move forward	Successful
Mounted “Slide Servo” to slide the fork 3 inches horizontally	The fork to slide	Successful
Mounted “Slide Servo” to lift the fork 3 inches vertically	The fork to lift	Unsuccessful. The servo would slide down but not up because it wasn’t strong enough; needed more torque
Modified the “Slide Servo” solution to lift the fork 3 inches vertically	The fork to lift	Successful
Read input switch	Read high and low correctly	Unsuccessful. Need to add pull-up/ pull-down resistor
Add the necessary resistors and read input switch again	Read high and low correctly	Successful

Testing the Software

We also tested the software portion of our design and performed the following tests shown in Table 7. Our code is can be seen in the appendix. Since the actual course will not be available until Spring of 2015, we made our own course. Figure 5 shows the course that we ran our software tests on.

Table 7: Software Test Table

Test Tried	What We Expected	Results
Change motor direction when switch is hit	Motor to change direction	Successful
Robot follows black line using sensor array	Robot stays on line	In Progress
If previous test doesn't work, calibrate code to correspond to sensors and then check to see if robot follows black line	Robot stays on line	In Progress
Robot slows down when sensors detect horizontal line	Robot slows down	In Progress
Servo drop off and pick up sequence for rings	Robot picks up and drops off rings	In Progress
Introduced slight offset; check servo drop off and pick up sequence for rings	Robot picks up and drops off rings	In Progress

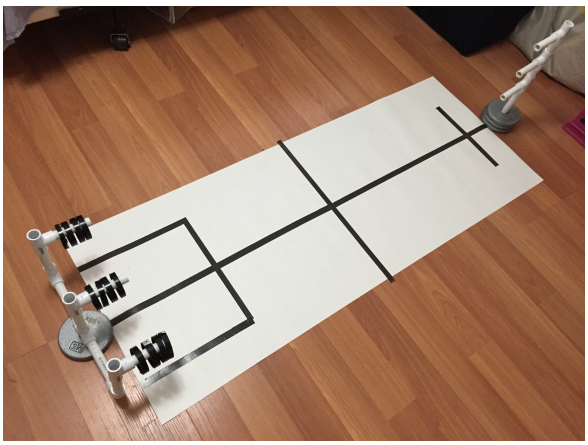

Figure 5: Self-made Course

CONCLUSION AND FUTURE WORK

Pl-x will be competing in Roborodentia during Cal Poly's Open House on April 18th 2015. In our minds, the design of Pl-x is built as efficient as possible while still operating within the given size constraints. The 12" fork is initially stored vertical to not exceed the 11" starting size constraints and is perfectly within the 13" operating size constraints when horizontally sliding. The two side prongs of Pl-x's fork are spaced 4" and 7" away from the center prong, which corresponds to the two set distances for the supply and scoring pegs. Therefore two pick-up or drop-off sequences must be executed for the respective side; once for two of the pegs and another for the remaining peg. Pl-x is designed to pick-up all the rings before moving across the field, therefore minimizing overhead travel time. To minimizing cost and moving parts, the L bend provides a height gain of 4" when the spin servo moves the fork from the supply side to the scoring side. In addition, the L bend gives Pl-x the ability to not have to turn around, further increasing efficiency. These design decisions should provide Pl-x a good fighting chance at this years Roborodentia.

Roborodentia gave our group the opportunity to work on an open ended design project. We received a better understanding of the design process and how necessary both brainstorming and physical prototyping were. During the beginning of the project, we made the mistake of jumping in and just ordering motors that worked for a similar project without properly doing calculations. Upon testing, the motors could not support the weight of our robot. We literally had to pay the cost and wait another week for new motors to ship in. There were also other times where we spent too much time discussing the feasibility of a solution, when we could have just went ahead and physically tried it. Our senior project also heavily reinforced Cal Poly's "Learn by Doing" motto, which pushes us to work on side-projects and not be afraid to break things. We also learned the importance of being open minded and never dismissing ideas without substantial consideration. Many of our game-changing ideas came about when one person's idea triggered another person's thought process.

There is still work that needs to be done on Pl-x. Our sliding fork solution did not work the way we expected; the servo's torque ended up being exerted on servo's mount rather than pushing and pulling the fork. Our current solution in the works involves a string and a spool to slide the fork. This solution would take up a substantially less space if we had a motor instead of a servo that only goes 180º. To further improve our design, we would remove the nails used for our fork's prongs and replace them with something lighter but just as rigid. Also if cost were not a factor, stronger servos could improve the speed and reliability. A distance sensor could also be used in place of our wall detection switches, allowing Pl-x to speed across the course and slow itself done as it nears the wall. With our design, we ended up ignoring the special objective of the peg in the center of the course. However, by swapping some the spin servo to a 360º servo and adding some additional code, we should be able to accomplish the bonus objective.

REFERENCES

"Arduino - PinMapping2560." Arduino - PinMapping2560. Arduino, n.d. Web. 06 Mar. 2015. <http://arduino.cc/en/Hacking/PinMapping2560>.

"Pololu - Arduino Library for the Pololu QTR Reflectance Sensors." Pololu - Arduino Library for the Pololu QTR Reflectance Sensors. Pololu, n.d. Web. 07 Mar. 2015. <https://www.pololu.com/docs/0J19/all>.

"Pololu - QTR-8A and QTR-8RC Reflectance Sensor Array User's Guide." Pololu - QTR-8A and QTR-8RC Reflectance Sensor Array User's Guide. Pololu, n.d. Web. 07 Mar. 2015. <https://www.pololu.com/docs/0J12/all>.

"Pololu - QTR-8x Reflectance Sensor Array with 11×1 Connection Pins Labeled." Pololu - QTR-8x Reflectance Sensor Array with 11×1 Connection Pins Labeled. Pololu, n.d. Web. 07 Mar. 2015. <https://www.pololu.com/picture/view/0J621>.

Reichenstein7. "Arduino Modules - L298N Dual H-Bridge Motor Controller." Instructables.com. Instructables, n.d. Web. 07 Mar. 2015. <http://www.instructables.com/id/Arduino-Modules-L298N-Dual-H-Bridge-Motor-Controll/>.

Seng, John. "Roborodentia 2015." Roborodentia 2015. N.p., 14 Jan. 2015. Web. 04 Mar. 2015. <https://docs.google.com/document/d/1hEpUtLgn5UAsiko7OFaVtgVSniHXmVvJ0WFP1OPBKhI/pub>.

"The Software Servo Library." Arduino Playground. Arduino, n.d. Web. 07 Mar. 2015. <http://playground.arduino.cc/ComponentLib/Servo>.

APPENDIX OF CODE

```
/*****************************************************************************
 File:
 Names: Kelly Leung, William Luo, and Jeffrey Tang

 Date: 3/5/15
 Description: This is the code that interprets switches and sensor inputs to
 drive the motors and servos
*****************************************************************************/

/*****************************************************************************
 Libraries
*****************************************************************************/
#include <QTRSensors.h>

/*****************************************************************************
 Constants
*****************************************************************************/
#define DEBUG         1

#define NUM_SENSORS   8     // number of sensors used
#define TIMEOUT       2500  // waits for 2500 microseconds for sensor outputs to
go low
#define EMITTER_PIN   2     // emitter is controlled by digital pin 2
#define toSupplyA     4
#define toScoreA      5
#define toScoreB      6
#define toSupplyB     7
#define speedPinA     8
#define speedPinB     9

#define WHITE_BLACK_BOUNDARY  400

#define A_DEFAULT_SPEED  255
#define B_DEFAULT_SPEED  255
#define SPEED_INTERVAL   50

#define SUPPLY_TRIGGER   23
#define SCORE_TRIGGER    52

#define MOVE_STATE       1
#define NO_MOVE_STATE    0

//Sensors 0 through 7 are connected to digital pins 33 35 37 39 41 43 45 47,
//respectively
//Constructor declares an object of sensors
QTRSensorsRC qtrrc((unsigned char[]) {24, 26, 28, 30, 32, 34, 36, 38},
  NUM_SENSORS, TIMEOUT, EMITTER_PIN);
```

```
//Arrays and variables
unsigned int sensorValues[NUM_SENSORS];
unsigned int state = MOVE_STATE;

unsigned char speedA = A_DEFAULT_SPEED;
unsigned char speedB = B_DEFAULT_SPEED;

//Runs once on boot up; defines what is input and output
void setup()
{
  pinMode(toSupplyA, OUTPUT);
  pinMode(toScoreA, OUTPUT);
  pinMode(toScoreB, OUTPUT);
  pinMode(toSupplyB, OUTPUT);
  pinMode(speedPinA, OUTPUT);
  pinMode(speedPinB, OUTPUT);
  pinMode(SCORE_TRIGGER, INPUT);
  pinMode(SUPPLY_TRIGGER, INPUT);

  if (DEBUG) {
    delay(500);
    Serial.begin(9600); // set the data rate in bits per second for serial data
transmission
    delay(1000);
  }

  //SetupFork();

  //testing
  //analogWrite(speedPinA, 70);
  //analogWrite(speedPinB, 70);
}

//Continuously runs; deals with everything - switches, running the motors, etc.
void loop()
{
  /*
  if (digitalRead(SUPPLY_TRIGGER) == high) {
    state = NO_MOVE_STATE;
    PickUpRings();
    digitalWrite(toSupplyA, LOW);
    digitalWrite(toScoreA, HIGH);

    digitalWrite(toSupplyB, LOW);
    digitalWrite(toScoreB, HIGH);
    state = MOVE_STATE;
  }
  else if (digitalRead(SCORE_TRIGGER) == high) {
    state = NO_MOVE_STATE;
    DropOffRings();
    digitalWrite(toSupplyA, HIGH);
    digitalWrite(toScoreA, LOW);
```

```
    digitalWrite(toSupplyB, HIGH);
    digitalWrite(toScoreB, LOW);
    state = MOVE_STATE;
  }
  */

  //TESTING
  digitalWrite(toSupplyA, HIGH);
  digitalWrite(toScoreA, LOW);

  digitalWrite(toSupplyB, HIGH);
  digitalWrite(toScoreB, LOW);
  //end testing

  //Read raw sensor values into sensorValue array
  qtrrc.read(sensorValues);

   speedA = A_DEFAULT_SPEED;
   speedB = B_DEFAULT_SPEED;

   if (state == MOVE_STATE) {
      if (sensorValues[0] < WHITE_BLACK_BOUNDARY) {
         speedA = speedA - SPEED_INTERVAL;
      }
      if (sensorValues[7] < WHITE_BLACK_BOUNDARY) {
         speedB = speedB - SPEED_INTERVAL;
      }
      analogWrite(speedPinA, speedA);
      analogWrite(speedPinB, speedB);
   }
   else {
      analogWrite(speedPinA, 0);
      analogWrite(speedPinB, 0);
   }

  //Print the sensor values as numbers from 0 to 2500, where 0 means maximum
  //reflectance
  if (DEBUG) {
    for (unsigned char i = 0; i < NUM_SENSORS; i++)
    {
      Serial.print(sensorValues[i]);
      Serial.print('\t'); // tab to format the raw data into columns in the
Serial monitor
    }
    Serial.println();
    delay(250);
    Serial.print("Speeds ");
    Serial.print(speedA);
    Serial.print(' ');
    Serial.print(speedB);
    Serial.println();
```

```
    delay(250);
  }
}

//Sub Functions
void SetupFork() {

}
void PickUpRings() {

}
void DropOffRings() {

}

//Sub Sub Functions
void Back5Inch() {

}
void Forward5Inch() {

}
void PerformPickSequence() {

}
void PerformDropSequence() {

}
```