The Story of Beyoncé:  
The Roborodentia 2016 Contestant

Created by:
Brandon Arnold
Manasa Kulkarni
Aaron Newberg
Jon Sleep
Problem Statement

Overview of Roborodentia XXI

Roborodentia is Cal Poly’s annual open house robotics competition. The event is a double elimination bracket tournament that assesses autonomous robots on their ability to score the most points according to the challenge specifications for that year. This year’s contest, Roborodentia XXI, based scoring on a robot’s ability to maneuver throughout the playing field (Figure 1) and pickup rings made of 2” diameter painted PVC pipe and place them onto scoring pegs. Points differed based on level of difficulty for the robot to pick up the rings and to access the scoring pegs. The robot that scores the most points within a three minute time period wins that round. Primary rings (red or blue) are exclusive to each robot and are the lowest scoring rings, worth 1 point when placed on Scoring Peg #1 and 3 points when placed on Scoring Peg #2. Scoring Peg 2 yields more points due to the ¾ inch obstacle wall placed in front of the pegs. Secondary rings are worth 2 points on Scoring Peg #1 and 8 points on Scoring Peg #2. Center rings (yellow) which are shared between both robots are worth 3 points on their own. More images of the course can be found in Appendix A.1. By naming our robot Beyoncé, we hoped that she, just like the world famous musician Beyoncé, would simply want to “put a ring on it”.

Figure 1: Course specification mock up - 8’ wide x 8’ long with 4” high walls surrounding the edges and along the center

1 A reference to the American singer-songwriter Beyoncé’s 2008 hit single “Single Ladies (Put A Ring On It)”

2 All course sketches can be found on the official Roborodentia Contest Rules through roborodentia.calpoly.edu
Competition Rules and Regulations

The complete and official competition rules are listed in Appendix A.2, however a simplified version of the rules and regulations can be found below.

1. The robot must move:
   The rules only allow for a 12”x12”x12” initial space for the robot and then to 14”x14”xunlimited for after start. Comparing this to the size of the course, it is basically imperative that the robot will need to move to be able to score.

2. The robot must follow lines:
   Because the robot is now moving, the robot must be able to follow the lines on the course to be able to successfully travel between the scoring and supply pegs.

3. The robot must be able to grab and drop rings:
   We must have some sort of mechanism for grabbing and dropping rings on the robot. It will be even better if we can grab multiple rings at once from a supply pegs.
   It will be even better than that if the robot can grab multiple rings from multiple supply pegs all at once, or at least sequentially.

4. The robot must be able to move over the rectangular bump (optional):
   If we want to score using the scorings pegs over the bump we will need to make sure that our robot will be able to traverse it. Placing rings on this peg is worth more points.
The Birth of Queen Bey

Our Design Process
We initially brainstormed ideas for the robot (crane, arms, forklift) and chose a few ideas. We then tried to design them through drawing/CAD and iterated through different versions. We tailored our design to try and fulfill the requirements listed in the problem statement.

The Different Ideas
Initially the idea was have the robot as a static arm that would just grab and drop rings and pivot from the shoulder place on center of the track. This was soon discovered to be impossible due to the sizing requirements. We then decided we would need motor with wheels and a linear motion controller that would make our robot into an emulation of a forklift. After deciding that linear motion controllers are too costly, we decided to go back to the arm design, but on top of the robot with wheels.

Dynamic design
We planned to do things in a certain way but changed them as necessary (such as adjusted length of the arms and the angles of the servos based on the accuracy of the drive train). We tried to keep hardware the same and change only the code as much as we were able to. We developed an arm testing program to figure out servo numbers and replaced the macro constants we were using in our main program with our tested values.

Software
In the deployed version of our code (that is, the codeset that controlled the robot during the competition), two running software threads cooperated to ensure operation of the robot. One of these threads was running on an Arduino Uno (ATmega328P microcontroller) and controlled all the servos which made up the arms of the robot. The other thread ran on an Arduino Mega (ATmega2560 microcontroller) and controlled the movement of the robot on the course as well as signalling the Arduino Uno when rings needed to be grabbed or placed.

Additionally, a testing program was written to allow individual control of the servos connected to the Arduino Uno so that we could determine the degree position that each servo should rotate to when either picking up from or placing rings onto the course’s pegs. This additional program was instrumental in the completion of the project as it was what allowed us to determine, in a timely manner.
manner, what the position of each servo should be without the need to upload different code onto
the microcontroller each time we wanted to test. By using this program we were able to find the
necessary degree-positions of each servo which we then stored as static arrays for reference in the
main program.
Below are two state transition diagrams which represent the operation of the robot. The first (Figure
2) is representative of the robot's operation as seen during the competition and trial runs. The
left column of states are those of the Arduino MEGA and the right column are the states of the
Arduino UNO. The second state transition diagram (Figure 3, found on the next page) shows the
operation of the testing program which allowed the individual control of the different servos
connected to the system.

![State transition diagram for deployed robot.](Figure 2)
Hardware
As mentioned in the section about software above, two microcontrollers (an Arduino UNO and an Arduino MEGA) served as the primary driving forces for our robot. The Arduino UNO had an Adafruit servo shield sitting atop it that allowed for I²C control of up to 16 servos. On top of the Arduino MEGA sat a shield designed and manufactured by Cal Poly alumnus Brian Gomberg and Cal Poly professor Dr. John Seng. This ‘RoboShield’ allowed for simplified interfacing with DC motors through the use of a library written by Mr. Gomberg and Dr. Seng and onboard motor encoders. Simply: the MEGA controlled the motion of the robot itself and the UNO controlled the motion of the servos and thus the arms of the robot. A full circuit diagram of the robot is available as an attachment to this document in Appendix D.

In addition to the electronic circuitry we cut two platforms on which the electronics would be mounted out of acrylic plates as well as acrylics pieces for the arms of the robot and for the grabbers. The CAD mockups used when laser cutting are described and displayed below.

CAD Designs
Base Plate
We designed a base on which would be mounted the MEGA w/ stacked RoboShield, battery, attachment for the IR sensor, and which would have mounting holes for the stand-offs to attach to the top plate. On the underside of this plate two motors and a single caster wheel were mounted to allow our robot to move. The CAD mockup for this is displayed on the next page in Figure 4.
Top Plate

We designed an upper plate that would be attached to 2” stand-offs connected to the base plate described above. On this upper plate when cut mounting holes for the Arduino UNO as well as holes with which to the large servos which served as the ‘shoulders’ of the robot. The CAD mockup for this plate is shown below.
**IR 3D Parts**

We 3D-printed a part of custom size that would house the IR Sensor array at the bottom of the robot where it would detect the black lines on the course which we used for navigation while the robot was running. This piece would screw into the robot base at its top, and has a gap in the center to cut costs (on 3D printer filament) and allow wires to conveniently travel through it.

![3D model of the IR sensor housing designed for the project](image)

*Figure 6: 3D model of the IR sensor housing designed for the project*

**Robot Arm and Grabber**

The robot arms were designed to allow for a wide range of motion for each arm so that rings could be reliably grabbed and placed with each of the 3 arms. Three arms were designed and built for the robot, but only one (the central arm) was mounted onto the robot by the date of the competition due to unforeseen difficulties associated with the control of the arms.

Each arm was comprised of several servos (3 for the central arm and 4 for the left and right arms) attached to acrylic spars. At the end of each of the arms was a “grabber” that was designed to grip the rings and allow us to score points during the competition. Shown on the next page is a 3D model of the final revision of the grabber that was used during the competition.
Three grabbers (one for each arm) were cut from the same acrylic plate that the top and bottom platforms and had additional spars attached to them to increase their vertical reach so that each grabber was able to make contact with a minimum of four of the course rings (2” spars since each ring was cut to be ½” tall). The grabbers are operated by attaching a servo to one arm of the grabber (which the servo has the ability to turn) and attaching the second arm of the grabber onto a machine screw such that its motion is also controlled by the servo. While the design for all laser-cut parts including the grabbers was done digitally, all design for the arms themselves was done on paper, and so no 3D mockups are available to be included in this section of the document.

**Choosing parts**

We chose parts directly based off the requirements. Seng had already recommended a central few - the arduino mega paired with the roboshield motor/servo controller he designed. He also had a deal for motors with encoders, that lead us to Pololu to buy mounts and wheels for the motors, as well as an IR sensor array for line detection. We bought a big sheet of acrylic for use with the chassis of the robot as well as the arms. Many servos were bought for the arms. Two 3000mAh batteries were brought to power the robot. See the full list in Appendix C.

**Robot Movement**

If you’d like to see our [unreadable] Arduino code, check out Appendix B.

**Simplified Line Following Function**

Position was a value between 0 - 7000 that was given to us from the IR sensor array and it’s API. The sensor values were up to 1000 value. We used the position value to create a proportional controller that would set either faster when it read it was off the line on the same side. This increase in speed would help account for the error off the line. This algorithm would also stop the robot.
once it got to an intersection was hit, at either a scoring location or a location where we needed to turn.

Main Driving Loop
The main loop our robot used to navigate between the supply and scoring pegs was as follows:

- Grab Rings
- Turn Around
- Follow Line until Intersection (middle)
- Turn Right
- Follow Line until Intersection (scoring fork)
- Place Rings
- Turn Around
- Follow Line until Intersection (middle)
- Turn Left
- Follow Line until Intersection (supply fork)
- Repeat

Arm Control
Each of the arms was controlled via a simple algorithm that is also described in the ‘Software’ subsection of the section detailing design of our robot. The Arduino MEGA necessarily tracks the location of the robot on the course and when it requires a specific action from the arms, it triggers an interrupt on digital pin 2 of the Arduino UNO. Depending on the UNO’s state of execution at the time that it receives the interrupt, it will either place the rings that it is holding down onto the pegs in front of the robot or grab the rings from the pegs in front of it.

In order to grab rings from or place rings on pegs, the UNO maintains a set of arrays containing the degree values which each servo in the system must be set to in order to grab the rings or place them. Once the degree values had been retrieved, the UNO accesses (via an I²C interface) the servo channel on the attached servo shield and sets the position of the servo attached to that channel by passing a PWM pulse width associated with the desired degree position. This is done for each servo and provides a quick and effective means of both grabbing and placing rings from or onto the course’s pegs.
Critic’s Response to Beyonce

Pre-Performance Issues (Problems we faced)
We soldered Dr. Seng’s RoboShield incorrectly… more than once.

● It was a mess choosing between male/female headers, we had to desolder and resolder multiple pins to figure out the correct configuration for the RoboShield
● Little did we know that Professor Seng published documentation on GitHub; If we had known this we would have been able to solder it correctly the first time

The button on the RoboShield was not working, probably as a result of our soldering/desoldering mess.

● We had to ask Professor Seng and he figured out the problem
● He found the issue; one of the resistors on the RoboShield was shorted out
● He helped us do a work-around by soldering a wire from the resistor directly to the lead on the button, and it worked!
● The button helped us greatly in future testing and was critical to us completing our goal

We measured the dimensions of our robot incorrectly, thanks to a ruler that interestingly used double-millimeters instead of regular millimeters.

● The first iteration of the 3-D printed IR sensor mount was twice as long as it should have been, due to this ruler
● The wood we used to mount the caster wheel was twice as long as it needed to be, so we had to chop it in half once we figured the ruler issue

At first, we had the motors and wheels screwed onto the acrylic board towards the front of the robot.

● After a sanity check with Seng, he told us that the motors should be in the middle of the robot so that it can turn in place without having to reverse
● This ended up making our robot much more reliable with turns on the course

In initial testing for line-following, we had to wave the IR sensor back and forth so that it could calibrate every time the robot started up. This was a huge pain.

● We later figured out that we could calibrate it once, extract the values that the IR sensor was using, and hard-code them into our program as constants
● This saved us a lot of time and prevented us from looking silly waving our robot back and forth every time Beyonce started up
We didn’t know if we would be able to supply enough power and current to the servos.

- Our battery produced enough voltage on its own, but it was doubtful that the amount of current available would suffice
- We went to Dr. Seng and he told us that his RoboShield’s servo ports would supply enough voltage for all of our servos and its current would suffice

In code, we initially tried to use pin-change interrupts to communicate between the Arduino Mega and the Uno.

- We couldn’t figure out the Arduino library that controlled these interrupts
- Ended up just sending signals from the MEGA to the UNO on state changes, this was much more simple, but it’s bad practice - and worked best for us at the time

We couldn’t develop a way to adjust the robot to align to the scoring pegs in time for the competition.

- Once the robot reached the scoring pegs after picking up rings from the supply pegs, we would have to be very lucky and be coincidentally correctly aligned to the pegs in order to score
- This led to our robot placing rings directly next to the scoring pegs, barely missing them

Our plan to use three arms didn’t happen in time for the competition.

- A slight delay in arm-building caused us to only have time to build and calibrate one arm
- If we had an extra day or two, we could have built all three arms and used them to score huge amounts of points per run. This was just poor planning on our part.

Sometimes the arm did not pick up all the rings or fell off of the robot.

- The arm would lose grip of a few rings and they would fall onto the course or back onto the supply peg from which it came
- This was more a mechanical issue than a coding issue because the plastic on the rings would easily slide off the grabber.
- After trying out multiple materials to attach onto the grabber to increase friction (this included tape, cardboard, melted glue, and cork, just to name a few), we found the perfect grip was the material of heavy duty anti slip work gloves. This prevented rings from falling out of the grabber.
- If the arm faced too much pressure in the wrong spot, a joint would come lose and the arm would fall onto the course
Performance review
The first few rounds of the competition were a little nerve racking. During our first round, the arm broke off when it was hit by a bad angle. Since we were seeded in double elimination bracket, we still had one more chance. Our second round was a win due to a forfeit. Before our third round we reattached the arm, but we were still nervous about our robot scoring. Beyoncé was doing well picking up rings, but when she reached the scoring pegs, she would always miss because of our lack of adjustment code. The rings would be placed just to the left of the scoring pegs instead of on them. By our last round, we figured this out and started Beyoncé just slightly off centered. Alas, we finally got one ring onto a peg. Below is a summary of how our robot performed.

Round 1: Loss
Round 2: Win (Forfeit)
Round 3: Loss

Regardless of the Roborodentia score, we actually accomplished our main goal. Even though some things didn’t go as expected, like the arm dismounting during the first round, the general goal of being able to pick up multiple rings with one grabber and place them down was accomplished. Unfortunately we didn’t calibrate the robot well enough before the start of the competition to get the rings placed in the right location. We also should have taken into consideration the line following algorithm which often faces inconsistencies based on lighting and shadows.

Future Endeavors
Although Roborodentia XXI is over, there is still room for improvement. If we could enter in Beyoncé again into next year’s competition we would hopefully have multiple arms in order to pick up three times as many rings and to perform more tests on the course. Along the lines of testing, we could always benefit from testing out the material before. Many of our issues had more to do with the mechanical properties of the the robot rather than the code. As computer engineers, we tend to focus more on the code and electrical components of the project. This experience taught us that even if our code is working, the mechanics can break, as did our arm during the first round of this year’s competition.

Since most of us are graduating and no longer eligible for Roborodentia, we also came up with other more applicable uses for Beyoncé. By removing the autonomous requirement of the contest, the physical robot can be controlled for other projects. The movement of the arm can be used to pick up and place other objects such as drinks. With stronger mechanics to support the weight of heavier objects the robot could be made into a robotic waiter that grabs drinks for people. Another idea was for an automatic pet feeder. With some additional components additional servos to account for other movements and an automatic can opener, the robot could be used to pick up cans of pet food, open it and turn it upside down to place the food in a dish. This mechanism if controlled wirelessly could be helpful to travelling pet owners who could remotely control when to feed their pets. These
ideas, though possible, would require a lot more work to be done on the physical robot, namely strengthening the mechanics, before implementing the additional features of control.
Appendices

Appendix A

A.1 Roborodentia Additional Course Images

Primary rings stacked on three pegs, exclusively available to the robot occupying that division of the course.

Obstacle en route to Scoring Pegs #2.

---

4 All course sketches can be found on the official Roborodentia Contest Rules through roborodentia.calpoly.edu
Secondary rings placed freely to left of Scoring Pegs #1.

Center Rings eligible for both robots.
A.2: Roborodentia 2016 Contest Rules

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.

---

5 Official Roborodentia Contest Rules through roborodentia.calpoly.edu
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

Robo.ino code (core driving logic implementation)

```cpp
#include <QTRSensors.h>
#include <RoboShield.h>
#include <RoboShield_Defines.h>
#include <Wire.h>
define rightMotor 2
#define leftMotor 3
#define ON_BLACK 250
#define NUM_SENSORS 8

define ON_RIGHT 500
#define SMOOTH_RIGHT 1500
#define ON_LEFT -500
#define SMOOTH_LEFT -1500

#define SIX_IN 505
#define THREE_IN 235

QTRSensorsRC qtr((unsigned char[]) {31, 33, 35, 37, 39, 41, 43, 45}, 8);
RoboShield rs(0);

int ledPin = 22;       // LED connected to digital pin 13

unsigned int calibratedMinimumOn[] =
(212,264,260,260,256,260,260,312); //{212,260,212,212,212,260,264,364};
unsigned int calibratedMaximumOn[] =
(4000,4000,4000,4000,4000,4000,4000,4000); //{4000,3856,3688,3076,2740,3584,4000,4000};

void setup() {
  if (rs.buttonPressed()) {
    rs.debuggingMode();
  }

  // put your setup code here, to run once:
  Serial.begin(9600);
  //pinMode(ledPin, OUTPUT);     // sets the digital pin as output

  qtr.calibratedMinimumOn = calibratedMinimumOn;
  qtr.calibratedMaximumOn = calibratedMaximumOn;
}

byte FollowLine(int position, unsigned int sensors[8]) {
```
int error = position - 3350;

int sLM = 19;
int sRM = -17;

static byte done = 1;
if (done) {
    done = 0;
    rs.resetEncoder(ENCODER_0_PIN);
    rs.resetEncoder(ENCODER_1_PIN);
}

if(error < 0) { // left of line, speed up left motor
    sRM -= ((double) abs(error) ) / 3350 * 10;
}
else if(error > 0) { // right of line, speed up right motor
    sLM += ((double) abs(error)) / 3350 * 10;
}

Serial.print("L: ");
Serial.println(sLM);
Serial.print("R: ");
Serial.println(sRM);

byte sensor_count = 0;

for(int i = 0; i < NUM_SENSORS; i++) {
    Serial.print("[");
    Serial.print(i);
    Serial.print("]=");
    Serial.print(sensors[i]);
    Serial.print(" ");
    if(sensors[i] > ON_BLACK)
        sensor_count++;
}
Serial.println();
if(sensor_count >= 3){
    Serial.println("Hit fork");
    sRM = 0;
    sLM = 0;
    rs.setMotor(rightMotor, sRM);
    rs.setMotor(leftMotor, sLM);
    done = 1;
    return 0;
}

//RIGHT == MOTOR 2 == ENCODER 0
//LEFT == MOTOR 3 == ENCODER 1
/if(rs.readEncoder(ENCODER_1_PIN) - rs.readEncoder(ENCODER_0_PIN) > 10) {
    //potential encoder correction
    sLM--;
}*/

rs.setMotor(rightMotor, sRM);
rs.setMotor(leftMotor, sLM);
return 1;
}

void driveDistance(int distance)
{
    while(rs.readEncoder(ENCODER_1_PIN) < distance ||
    rs.readEncoder(ENCODER_0_PIN) < distance) {
        rs.setMotor(rightMotor, -12);
        rs.setMotor(leftMotor, 13);
    }
    rs.setMotor(rightMotor, 0);
    rs.setMotor(leftMotor, 0);

    Serial.print("You're an idiot");
}

void turnLeft()
{
    rs.resetEncoder(ENCODER_0_PIN);
    rs.resetEncoder(ENCODER_1_PIN);

    unsigned int distance = 525;

    while(rs.readEncoder(ENCODER_1_PIN) < distance ||
    rs.readEncoder(ENCODER_0_PIN) < distance) {
        rs.setMotor(rightMotor, -20);
        rs.setMotor(leftMotor, -20);
    }
    rs.setMotor(rightMotor, 0);
    rs.setMotor(leftMotor, 0);
}

void turnRight()
{
    rs.resetEncoder(ENCODER_0_PIN);
    rs.resetEncoder(ENCODER_1_PIN);

    unsigned int distance = 525;

    while(rs.readEncoder(ENCODER_1_PIN) < distance ||
    rs.readEncoder(ENCODER_0_PIN) < distance) {
```c
rs.setMotor(rightMotor, 20);
rs.setMotor(leftMotor, 20);
}
rs.setMotor(rightMotor, 0);
rs.setMotor(leftMotor, 0);

void turnAround() {
    rs.resetEncoder(ENCORDER_0_PIN);
    rs.resetEncoder(ENCORDER_1_PIN);

    unsigned int distance = 1220;

    while(rs.readEncoder(ENCORDER_1_PIN) < distance ||
    rs.readEncoder(ENCORDER_0_PIN) < distance) {
        rs.setMotor(rightMotor, 20);
        rs.setMotor(leftMotor, 20);
    }
    rs.setMotor(rightMotor, 0);
    rs.setMotor(leftMotor, 0);
}

void loop() {
    static byte position = 0;
    if(rs.buttonPressed() == 1) {
        unsigned int sensors[8];
        while(1) {
            // grab rings
            rs.lcdClear();
            rs.lcdPrintf("Grabbing Rings.");
            delay(1500);
            digitalWrite(INTERRUPT_PIN, HIGH); // sets the LED on
            delay(1);
            digitalWrite(INTERRUPT_PIN, LOW); // sets the LED on
            delay(3000);
            rs.lcdClear();
            rs.lcdPrintf("Turning around");
            turnAround();
            delay(500);
            rs.lcdClear();
            rs.lcdPrintf("Following to center");
            while(FollowLine(qtr.readLine(sensors), sensors)) {
                
            }
            delay(500);
```
rs.lcdClear();
rs.lcdPrintf("Turning right");
turnRight();
delay(500);
rs.lcdClear();
r
while(FollowLine(qtr.readLine(sensors), sensors)) {
    
}
delay(1000);  
//place rings
digitalWrite(INTERRUPT_PIN, HIGH);   // sets the arm on
delay(1);
digitalWrite(INTERRUPT_PIN, LOW);
delay(3000);
rs.lcdClear();
rs.lcdPrintf("Turn around");
turnAround();
delay(500);
rs.lcdClear();
r
while(FollowLine(qtr.readLine(sensors), sensors)) {
    
}
delay(500);
rs.lcdClear();
r
rs.lcdPrintf("Turning left");
turnLeft();
delay(500);
rs.lcdClear();
r
rs.lcdPrintf("Following to fork");
while(FollowLine(qtr.readLine(sensors), sensors)) {
    
}
}
### Bill of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Price</th>
<th>Quantity</th>
<th>Total Price</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Mega</td>
<td>$13.19</td>
<td>1</td>
<td>$13.19</td>
<td>Amazon</td>
</tr>
<tr>
<td>Roborodentia Motor</td>
<td>$10.00</td>
<td>2</td>
<td>$20.00</td>
<td>Robotics Club</td>
</tr>
<tr>
<td>Roboshield</td>
<td>$35.00</td>
<td>1</td>
<td>$35.00</td>
<td>Robotics Club</td>
</tr>
<tr>
<td>Acrylic 24&quot; x 18&quot; x 0.2 &quot; sheet</td>
<td>$22.65</td>
<td>1</td>
<td>$22.65</td>
<td>Home Depot</td>
</tr>
<tr>
<td>Laser Cutting / 3D Printing</td>
<td>$24.00</td>
<td>1</td>
<td>$24.00</td>
<td>D-Fab Cal Poly</td>
</tr>
<tr>
<td>Venom 7.2V RC Battery</td>
<td>$19.49</td>
<td>2</td>
<td>$38.98</td>
<td>Amazon</td>
</tr>
<tr>
<td>4-40 Screws 100 cnt</td>
<td>$2.14</td>
<td>1</td>
<td>$2.14</td>
<td>Amazon</td>
</tr>
<tr>
<td>4-40 .25&quot; Hex Standoff (10)</td>
<td>$7.35</td>
<td>1</td>
<td>$7.35</td>
<td>Amazon</td>
</tr>
<tr>
<td>4-40 2&quot; Hex Standoff (5)</td>
<td>$9.07</td>
<td>1</td>
<td>$9.07</td>
<td>Amazon</td>
</tr>
<tr>
<td>Tenergy Smart Charger</td>
<td>$15.99</td>
<td>1</td>
<td>$15.99</td>
<td>Amazon</td>
</tr>
<tr>
<td>4-40 Hex Nut (100)</td>
<td>$2.42</td>
<td>1</td>
<td>$2.42</td>
<td>Amazon</td>
</tr>
<tr>
<td>Pololu Universal Aluminum Mounting Hub for 6mm Shaft, #4-40 Holes (2-Pack)</td>
<td>$7.95</td>
<td>1</td>
<td>$7.95</td>
<td>Pololu</td>
</tr>
<tr>
<td>Machine Screw: #4-40, 1/2&quot; Length, Phillips (25-pack)</td>
<td>$0.99</td>
<td>1</td>
<td>$0.99</td>
<td>Pololu</td>
</tr>
<tr>
<td>Pololu Stamped Aluminum L-Bracket Pair for 37D mm Metal Gearmotors</td>
<td>$7.95</td>
<td>1</td>
<td>$7.95</td>
<td>Pololu</td>
</tr>
<tr>
<td>Machine Screw: #4-40, 1/4&quot; Length, Phillips (25-pack)</td>
<td>$0.99</td>
<td>1</td>
<td>$0.99</td>
<td>Pololu</td>
</tr>
<tr>
<td>Scooter/Skate Wheel 70×25mm - Black</td>
<td>$2.49</td>
<td>2</td>
<td>$4.98</td>
<td>Pololu</td>
</tr>
<tr>
<td>Pololu Aluminum Scooter Wheel Adapter for 6mm Shaft</td>
<td>$4.95</td>
<td>2</td>
<td>$9.90</td>
<td>Pololu</td>
</tr>
<tr>
<td>Pololu Ball Caster with 1” Plastic Ball and Ball Bearings</td>
<td>$6.95</td>
<td>2</td>
<td>$13.90</td>
<td>Pololu</td>
</tr>
<tr>
<td>QTR-8RC Reflectance Sensor Array</td>
<td>$9.95</td>
<td>1</td>
<td>$9.95</td>
<td>Pololu</td>
</tr>
<tr>
<td>Gorilla Glue &amp; Tape</td>
<td>$8.57</td>
<td>1</td>
<td>$8.57</td>
<td>Home Depot</td>
</tr>
<tr>
<td>Pololu Shipping and Handling</td>
<td>$5.95</td>
<td>1</td>
<td>$5.95</td>
<td>Pololu</td>
</tr>
<tr>
<td>16-Channel 12-bit PWM/Servo Shield (I2C interface)</td>
<td>$17.50</td>
<td>1</td>
<td>$17.50</td>
<td>Adafruit.com</td>
</tr>
<tr>
<td>High Torque Metal Geared servo</td>
<td>$19.95</td>
<td>4</td>
<td>$79.80</td>
<td>Adafruit.com</td>
</tr>
<tr>
<td>Micro Servos</td>
<td>$5.95</td>
<td>10</td>
<td>$59.50</td>
<td>Adafruit.com</td>
</tr>
<tr>
<td>Shipping for Adafruit items</td>
<td>$27.10</td>
<td>1</td>
<td>$27.10</td>
<td></td>
</tr>
<tr>
<td>Rubber Gloves</td>
<td>$9.88</td>
<td>1</td>
<td>9.88</td>
<td>Home Depot</td>
</tr>
</tbody>
</table>
Appendix D - Hardware Schematic