Final Design Report

2018 IDC

Section 01 Weightlifting

Justin Zwiebel & Tong Fu

To uphold the Duke Community Standard:

I will not lie, cheat, or steal in my academic endeavors;

I will conduct myself honorably in all my endeavors; and

I will act if the Standard is compromised. *Tong Fu & Justin Zwiebel*

I. Abstract

This year's Integrated Design Challenge required a team of five robots to perform Olympic themed sensing and calculation. Each bot was designed and built by a groups of 2-3 students from a section of the ECE 110 labs. The groups had BOE-Bots with an Arduino Mega 2560, basic electrical components in the ECE 110 lab, various sensors that could be requested, and their skills learned over their labs and lectures in order to successfully complete the challenge. The challenge consisted of each bot following a black ring around the competition surface. Once the bot completed a full circle, the bot could use whatever method the group deemed best to complete the sensing task given. Once each bot made its measurement, the bots had to communicate its measurement, 0,1, or 2, to the other bots. The bots had to add the all the measurements together, take the modulus three, and display the remainder through a song (2), light show (1), or dance (0). Our group, weightlifting, had to determine which weight was placed in the center of the ring. The three different weights differed in color and magnetic strength, and each corresponded to a value of 0,1, or 2. Our team successfully completed this challenge, with only a couple of mistakes for the entire team in the final demo. Our Bot only had one mistake, a miscalculation of the final remainder. The goal of this challenge is to introduce students to the engineering design process, from start to completion. Overall, the IDC challenge was successfully completed by our team.

II. Introduction

This year's Integrated Design Challenge requires five autonomous bots to act as judges for the BOE-lympics. Each individual bot first has to follow a black ring around their object to be sensed. The bots then must use sensors to take measurements such as color, size, slope, distance, and magnetic strength of an object. Based on the measurements taken, each bot will then transmit a 0, 1, or 2 to the each other bot based on what object it senses. The bots will add together the values it receives, in addition to its own value, and take the modulus of 3 of that number. When the remainder is a 2, the bots have to perform a song using a piezoelectric speaker. When the remainder is 1, the bots will perform a light show using LEDs or an LCD display. When the remainder is 0, the bots will perform a dance using servo motors.

Group 1, the diving group, is required to sense different objects that vary in size and color. Group 2, weightlifting, has to sense 3d printed "weights" that varied in size, color, and magnetic strength. Group 3, cycling, must sense different ramps that vary in steepness and color. Group 4, basketball, has to determine how far a "basketball", which is really an RFID chip, is placed from the hoop. Group 5, court sports, has to determine the playing surface in the center, each with a different color. The goal of this challenge is for each group to think critically on how to best attack its individual task, and then relay that information to the team as a whole. Each individual group has to determine what sensor, such as an RGB, QTI, accelerometer, hall effect, or RFID, will best suit its own task. Each group has to determine how to integrate this sensor on the BOE bot, in a way that would lead to accurate, repeatable, and reliable measurements.

Our group's bot is the weightlifting bot. The three objects the bot could have to sense is a Bronze medal weight, which is white and has no magnet, the Silver medal weight, which is gray and has a small magnet, or the Gold medal weight, which is black and has a large magnet. The

bot first must use QTI sensors to provide information to the Arduino, which will communicate to the servo motors and follow the ring around the course. Once the bot completes a circle, it must orient itself and move close to the weights, so that the Hall-effect sensor, a sensor that measures the strength of a magnetic field, can work effectively. Once the bot makes its measurement, it must save this data for the calculation later, and then use an XBee module to communicate with the other bots. It will send the value it senses, and listen for the values that other groups send. Once it receives a message from each group, the bot will calculate the remainder of the sum of the numbers it receives, and then perform its song, light show, or dance.

This challenge has many real world applications. The proliferation of autonomous vehicles and bots, in factories, in warehouses, and on roads, will require an extensive amount of design and planning. For autonomous vehicles to work effectively, they must communicate with each other, and sense changes in its environment, with minimal human input. For example, factories could employ autonomous robots that can detect problems in production, and then alert people or other robots to come fix the problem. In warehouses, autonomous robots can locate and deliver products efficiently, cutting down the time necessary for delivery. Autonomous cars need to sense varying road conditions, and adjust its parameters accordingly, so that they do not drive off the road. Overall, the skills and planning required for this challenge have many real world applications.

III. Experimental Procedures and Results

March 5–March 9:

Our group finalized the conceptual design report. The conceptual design report includes trade study, cost estimate, and a draft schedule. The trade study detailedly compared the cost, size, reliability, accuracy, and complexity for five possible sensors that can be used the weightlifting task. After comparing the five potential sensors, our group decided to use the Hall-effect sensor as it was the most reliable sensor and can be easily implemented onto the BOE-Bot. We used Smartsheet, and online work management website, to plan out the schedule and Gantt chart for the rest of the semester. For communication, we used an Xbee module (Figure 1) that allowed our bot to send and receive messages when a button was pressed. We wired two LEDs to show that the bot can transmit, receive, and display information. We observed that our bot worked well in sending and receiving messages.

March 19–March 23:

The goal for this week was for the bot to successfully follow a black line on the ground. We coded the bot so that it could follow four different types of lines: a circle, a square, a straight line with hash marks, and a sharply curved track. This task was accomplished by using the QTI sensors that converted an analog signal into a digital reading of black or white in the Arduino. We first decided to use four QTI sensors. The QTIs were tightly mounted with ⅞-inch screws, unthreaded spacers, threaded standoffs and ⅜-inch screws as Figure 2 indicates. After the array of sensors was implemented, our group built the QTI circuit based on Figure 3. After the mechanical part was done, we coded the line following task in Arduino. The main function "RCtime" measures how long it takes the capacitor to discharge through the IR resistor. This

function is shown in the final bot code in the Appendix. The Arduino charges the capacitor in the QTI, and then measures the time it takes for the RC circuit to decay. The phototransistor affects this time, and thus allows the QTI and Arduino to differentiate between light and dark.

Inside the loop, each sensor constantly called the RCtime function as the bot moves forward. After some tests, our group decided to reduce the amount of QTI sensors to three to simplify the program. Four sensors were difficult to arrange and code a proper threshold time in the circuit. While three sensors, did not allow for finely tuned movements, the bot reliably navigated all four courses. The biggest challenge we faced when we were coding for the line following was that the threshold for RCtime function changed often. We had to test the RC time frequently, as the wrong threshold value would render the circuit useless. Sometime the flickery threshold was due to QTI cables coming off the pinhole. After we set up a relatively stable threshold number, we divided the sensing scenarios into

(1) only middle sensor sensed black

(2) only left sensor sensed black

(3) only right sensor sensed black

(4) the right and middle sensor sensed black

(5) the left and middle sensor sensed black

(6) all sensors sensed black

For case (1), the bot should keep going, and the code for moving bot forward is

```
servoLeft.writeMicroseconds(1550);
servoRight.writeMicroseconds(1450);
```

For case (2), the bot should turn left, and the code for turning bot to the left is

```
servoLeft.writeMicroseconds(1490);
servoRight.writeMicroseconds(1420);
```

For case (3), the bot should make a right turn, and the code for turning bot to the right is

```
servoLeft.writeMicroseconds(1580);
servoRight.writeMicroseconds(1510);
```

For case (4), the bot should make a sharp right turn, and the code for turning bot sharply to the right is

```
servoLeft.writeMicroseconds(1580);
servoRight.writeMicroseconds(1500);
```

For case (5), the bot should make a sharp left turn, and the code for turning bot sharply to the right is

```
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1420);
```

For case (6), the bot should stop, and the code for stopping the bot is:

```
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1500);
```

In this case, the bot should also keep moving after it stops for 3 seconds for the hash mark task, and the code for doing this is

```
servoLeft.writeMicroseconds(1500);
```

servoRight.writeMicroseconds(1500);

delay(3000);

servoLeft.writeMicroseconds(1530);

servoRight.writeMicroseconds(1470);

delay(500)

During the lab demo, our bot followed all of the lines correctly. The bot also can distinguish the squared line and the straight line with hash mark very well, although these two lines both have a section that is all black which made it harder for the bot to execute the code and know if it's going to turn or keep straight after stopping for 3 seconds. Additionally, cases 4 and 5 were subsequently removed when only the circle needed to be followed since those cases never occurred in that scenario.

March 26-March 30:

Our goal for this week was to be able to sense the correct object, send a message to other bots, and receive a message from other bots. We added the hall effect sensor to the bot to be able to sense different strengths of magnetic fields. The hall effect sensor works by driving a current across a conductive plate. In the presence of a magnetic field, a force acts on the moving charges, which moves the electrons to one side of the plate, creating a negative charge, and leaving holes in protons on the other side of the plate, creating a positive charge. The sensor connects a wire to each side of the plate, and then runs those wires into an op amp. The potential difference is amplified and outputted as a voltage (Figure 4). This voltage can read by the analog pin on the Arduino, and the code calculates the absolute value of the strength of the magnetic field (because

the polarity may change based on the orientation of the field). Based on the value that the Arduino measures and then calculates, the bot utilizes one LED to display the object it senses. Similarly, we also had to set up two stable and reliable thresholds. The upper limit and lower limit we decided to use in our final bot code were 1600 and 180.

Inside the main loop, we divided the sensing scenarios into

(1) the absolute value of the output was greater than 1600

(2) the absolute value of the output was between 180 and 1600

(3) the absolute value of the output was less than 180

For case (1), the Hall-effect sensor senses a stronger magnetic field, which indicates a black kettlebell is placed in front of it. In this case, the red LED shows a stable signal. For case (2), the Hall-effect sensor senses a less stronger magnetic field, which indicates a grey kettlebell is placed in front of it. In this case, the red LED flickers. The code for this case is

```
If (abs(mag)) > 180 && abs(mag) < 1600) {
digitalWrite (11,HIGH)
delay(300)
digitalWrite(11,LOW)
}
```

For case(3), the sensor senses no magnetic field (or very little). Therefore, a white kettlebell with no magnet is placed in front of the bot. The LED will not light up.

We implemented the sensor directly on the BOE-shield breadboard instead of in the front or back of the bot, because we could manually control the location of the bot and target object when

sensing for this week. We observed while testing that the sensor's readings were highly dependent on orientation, distance, and connection to the board. If the sensor was not properly connected to the wires, or even shifted slightly, the reading would come out differently. In addition, the orientation of the sensor to the magnet and the distance would drastically change the reading. For example, for the large magnet, the bot could be at the same difference, but if oriented improperly would read the there was no magnet. Using this information, we moved the sensor to the back of the bot, so it could stick out on L-Brackets and effectively crash into the target object, resulting in a consistent distance and orientation to make its reading. We also found that using only one LED is not straightforward enough in displaying the message the bot sent or received, and added a second, green LED. Figure 5 shows how the Hall-effect sensor is integrated into the bot.

April 2-April 6:

This week required us to integrate the line following, sensing, and communication code into one file. Now, the bot had the ability to perform all the necessary steps of the challenge. The first step was to rewire the bot. The new location for each pin for the QTIs, LEDs, Xbee module, and Hall-effect sensor is in the final code in the appendix. Instead of randomly assigning pins and breadboard space, the bot now had a much more methodical and neat wiring. We utilized one breadboard on the bot for the Xbee module and Hall-effect sensor, and the other breadboard for the QTI sensing and LEDs. This week required a lot of troubleshooting in order to integrate the code properly. Delays, not declaring pins properly, or utilizing the same pin for multiple actions led to inaccuracies in the performance of the bot. Overall, we were able to successfully troubleshoot all the difficulties in this section, and the bot performed successfully.

In order to improve the simplicity of the bot's code, the code was broken into different sections. A condition variable prevented communication from occurring until the bot had completed its line following, and prevented the line following code once the bot had completed the course. When conditional number equals to 0, the bot is in the line following mode. We simply add

```
&& (cond == 0)
```

to the line following code listed above. If the conditional number equals to 1, it means that the bot is now in the sensing mode. It will thus begin to send signals (numbers) to other bots. We did this by:

```
if (cond == 1){
If (abs(mag) > 2200){
digitalWrite(3,HIGH);
Char outgoing = '3';
Serial2.print (outgoing)
.... ...
}
}
```

After we revised the code for the sensing part, we had to code the motion part. This week's biggest challenge was that the bot could not turn accurately a 90 degrees angle when it met the hash mark. The magnetic field sensing highly depends on the distance between the sensor and the object, so if the turning angle is not precise enough, the bot will end up going to a deviated location, which would thus causing wrong readings. We then figure out that if we let the bot

moved slightly forward when it passes the mark, it could turn in a more precise angle. The code for this part is shown as following:

```
else if ((tL > t) && (tM > t) && (tR > t) && (cond == 0)) {
//all three see black
cond = 1; //changes condition so bot can run through other sections of code
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1500);
//bot stops
delay(500);
servoLeft.writeMicroseconds(1520);
servoRight.writeMicroseconds(1480);
//bot inches forward slightly
delay(400);
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1500);
//bot stops
delay(500);
servoLeft.writeMicroseconds(1450);
servoRight.writeMicroseconds(1450);
//90 degree turn
delay(1000);
servoLeft.writeMicroseconds(1470);
```

servoRight.writeMicroseconds(1530);

//reverse towards target (due to placement of sensor on back of bot)

delay(1950);

servoLeft.writeMicroseconds(1500);

servoRight.writeMicroseconds(1500);

//stop

delay(1000);

}

Lastly, the button was removed as the integrated code now handled message sending and receiving.

April 9-13:

The team had to correctly sense each individual object, and then transmit this information to the other bots. The main challenge that needed to be overcome was how each group should transmit information for the object it sensed, without overlapping with other groups. The solution was for each group to use its own set of three characters (1,2,3; 4,5,6; a,b,c; d,e,f; g,h,i) that corresponded to bronze, silver, or gold. The Arduino software would convert the message it received to a 0,1, or 2. We did it by converting the incoming characters into integers that can be used in future calculation.

if (Serial2.available()) {

char incoming = Serial2.read(); //sets incoming variable as that read by XBee

//Serial.println(incoming); (troubleshooting)

```
//This section converts the incoming character to an integer:
//Each if statement determines which range the message came from (1-3, a-c,d-f,g-i)
//what it finds the correct location, it assigns the x variable to a 0, 1, or 2 based on the
message received
//it flips the c variable to a 1 so that it stops listening for that groups message
//it plays a tone to let us know it has received a message

if ((incoming-48 < 4) && (incoming-48 > 0) && (c1 == 0)) {
x1 = incoming-49;
c1 = 1;
tone(4,220,200);
Serial.println(x1);
Serial.println(c1);
}
else if ((incoming-96 < 4) && (incoming-96 >0) && (c3 == 0)) {
x3 = incoming-97;
c3 = 1;
Serial.println(x3);
tone(4,220,200);
//Serial.println(c3);
… ...
}
```

By using separate characters for each group, the bot could ensure that it only counted each other group once. Condition variables, c1-c5, were used to ensure that the bot received a message from each group before moving on to calculations. Also, we added a piezoelectric speaker this week that buzzed when the bot received a message, and played different tones when the bot changed from different parts of its code. This addition eased and improved troubleshooting the bot. For the calculation part, we added a conditional number (cond==2) to tell the bot that it can start calculating the remainder when it received all messages from all bots.

```
if((c1 == 1)  && (c2 == 1)  && (c3 == 1)  && (c4 == 1)  && (c5 == 1) && (cond ==1)) {
cond = 2; //changes to final demonstration condition
total = (x1 + x2 + x3 + x4 + x5) % 3; //calculates remainder
outgoing = outgoing;
```

The bot was now constantly sending characters when it started to sense, and only received signals one time from each of the other bots. It would also calculate the remainder, and was prepared for the final tasks: perform differently based on the remainder.

April 16-20:

The final task for our group and the team was to complete the challenge by performing a song, dance, or light show based on the calculated remainder. We implemented code that correctly completed the challenge by performing a song, light show, or dance based on the messages it received and final calculation.

If the remainder is 0, the bot should be able to perform a coordinated dance. Our bot dances back and forth by executing this part of the code:

```
if(total == 0) { //bronze metal, bot dances back and forth

digitalWrite(6,HIGH); //displays green light
servoLeft.writeMicroseconds(1550);
servoRight.writeMicroseconds(1450);
Serial2.print(outgoing);
delay(1000);
servoLeft.writeMicroseconds(1450);
servoRight.writeMicroseconds(1550);
delay(1000);
Serial2.print(outgoing);
}
```

If the remainder is 1, the bots should perform a light show. As mentioned above, the drawback of having only one LED is that it is hard to tell if the bot is sensing, receiving or performing. We then decide to use three LEDs. This third LED separated the display of what the bot sensed, and the light show to be performed when the remainder was 1. In this case, the red and green LED will flash alternately.

```
else if(total == 1) {
```

```
//plays light show by flashing each light
digitalWrite(3, HIGH);
digitalWrite(6,LOW);
Serial2.print(outgoing);
delay(500);
digitalWrite(6,HIGH);
digitalWrite(3,LOW);
delay(500);
Serial2.print(outgoing);
}
```

If the remainder is 2, the bots are supposed to sing a synchronized song together. The code for the song is provided in the appendix. We also added

```
Serial2.print(outgoing);
```

into the performing section code to ensure that the bot would keep sending signals in case other bots didn't receive the message.

One final hardware addition was a provided weight. The additional weight provided greater traction for the wheels when the bot turned to face the weight it needed to sense. One final software change removed the delays from the communication code. While the delays were necessary to not bombard the Serial channel with only our information, it often prevented our bot from "hearing" other groups messages, as it was stuck in the delay portion of the code most of the time. The addition created two new variables. On each loop, the bot would count the time that had passed, in milliseconds. If the previous time subtracted from the current time was

greater or equal to a given interval, the bot would run through the communication sending code. This would provide greater "listening" time so that the bot could more quickly hear the other groups' messages.

Lastly, each group had to present a weekly status update that explained what they did that week, what went well, what went poorly, and why they made certain design choices. This ultimately culminated in a final oral defense of the entire design of the bot. Each group had to demonstrate a detailed understanding of their code, design, and function of their sensors.

IV. Analysis and Discussion

Overall, the Weightlifting BOE-Bot and team in general performed well in completing the necessary tasks. The team made very few mistakes in the final demo, with one incorrect sensing and a couple of miscalculations of the final remainder. In the final demo, our bot correctly completed all the tasks, except for one trial in which some groups calculated a final remainder that was different from other groups. Since this problem only occurred once, and the calculation and communication codes were almost entirely the same, the team determined that a bot may have sent the wrong character either before the trial started, or during the trial. In either case, this problem did not persist and was seen as an anomaly in the test as a whole. While the weightlifting bot performed well in sensing, there were specific areas that could have been improved, given more time. The biggest challenge was getting the hall effect sensor in the exact same orientation and location on each trial. Small deviations in the direction the bot faced and the distance of the bot to the target resulted in significant enough errors to lead to an incorrect reading. Although this was not a problem during the trial, it required a significant amount of preparation to fine tune the bot's turn and movement towards the target. One cause of this problem was the angle the bot faced when it reached the hash, signifying the turning point. The bot would either reach the hash with the QTI sensors parallel to the hash, or at a slight angle to the hash. This small difference was enough for the bot to turn an improper amount, and thus not line up correctly with the target. One method to rectify this problem would be a fourth QTI sensor, in addition to finer adjustments in the code, that would ensure the bot would reach the hash with the same orientation each time. Another, better solution would be to not hardcode the turn. Instead, an set of infrared LEDs and infrared sensors could allow the bot to navigate toward the object and stop at a predetermined location each time. This solution would also allow the bot

to better handle instances in which problems with the wheels or board led to the bot getting stuck on its turn and not completing the turn properly. More improvements, in terms of resources, would be to utilize multiple QTI sensors and hall effect sensors, and choose only the ones that performed the most reliably. Some design changes that did not cause problems but would have improved the overall look of the bot would be to use screws to hold the L-Brackets that held the hall effect sensor instead of tape, and snipping wires to the necessary length, instead of sticking out so much.

Parallax Shield Bot (*1)	$179
XBee (*1)	$28
Jumper wires(including 3-pin headers), & LEDs (*3), resistors	$10
QTI sensors (*3)	$30 ($10 * 3)
Hall-effect sensor (*1)	$1
Buzzer	$3
Total	$251

Table 1: Total Cost

While cost was not part of the design challenge, our bot did a successful job minimizing the total cost. Increasing costs could indeed increase the reliability of the bot though. By adding another QTI sensor, the bot could more accurately follow the line. Alternate designs with two Hall-effect sensors separated by some distance or a set of infrared “eyes” could improve the reliability of the reading. Ultimately though, these considerations were not necessary as our bot performed very well in the final demonstration.

V. Conclusion

Overall, the team successfully met the objectives of the challenge in the final demo, and the Weightlifting bot performed very well. The first objective, line following, was completed perfectly by every bot; although, our observations found that either an additional QTI sensor in addition to refined driving code would provide more reliable line following. Still, this did not pose a problem in the final demo. The second objective, sensing, was incorrect in only one bot for one trial. The Weightlifting bot sensed the correct object in each trial. Communication worked well for all groups. Every group sent and received the proper number of messages for each trial, so that a remainder could be calculated. In only one trial, the bots each calculated a different remainder and some performed the song, and some a light show. While we could not determine the cause of this issue, whether it be a sensing, communication, or calculation mistake, it did not persist and the team performed correctly in later trials. Lastly, the team successfully completed all parts of the engineering design process. Skills each group gained were prototyping and design, troubleshooting, and explaining the function and design of their bot. Overall, the team met all the objectives of the IDC challenge.

VI. Appendix

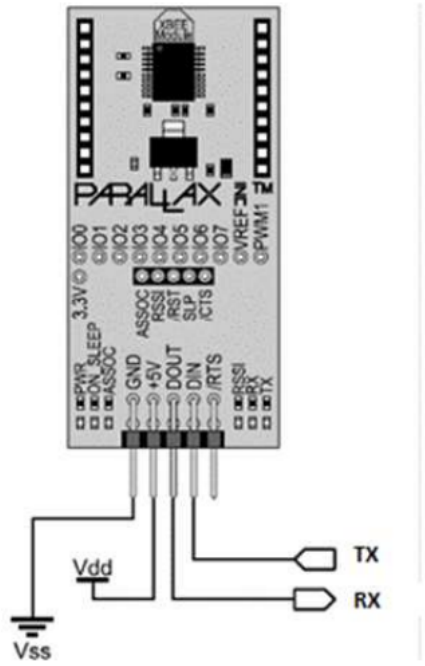

Figure 1: Parallax XBee Module

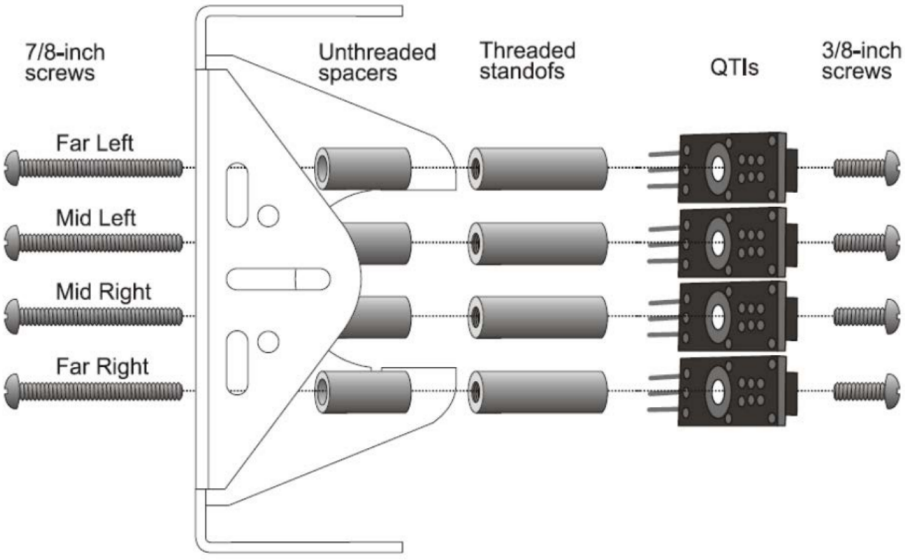

Figure 2. QTI layout (final design only used 3 QTI sensors)

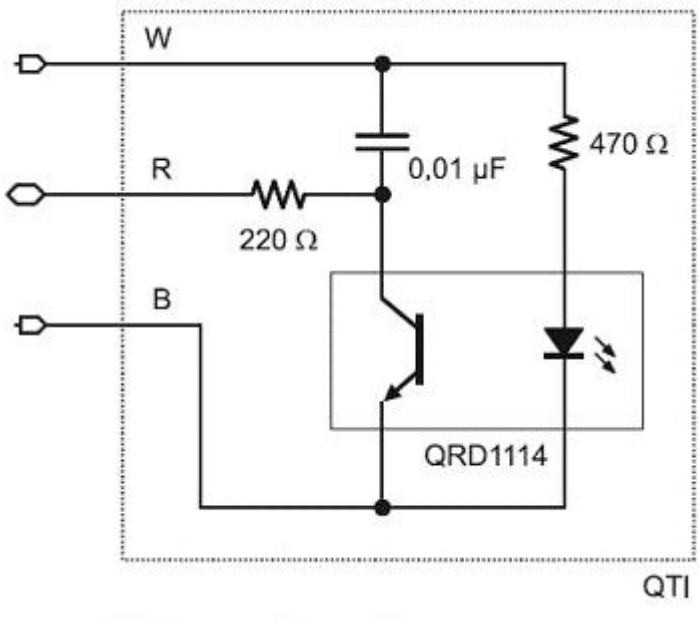

Figure 3. W connects to 5V, R connects to the Arduino pin, and B connects to ground.

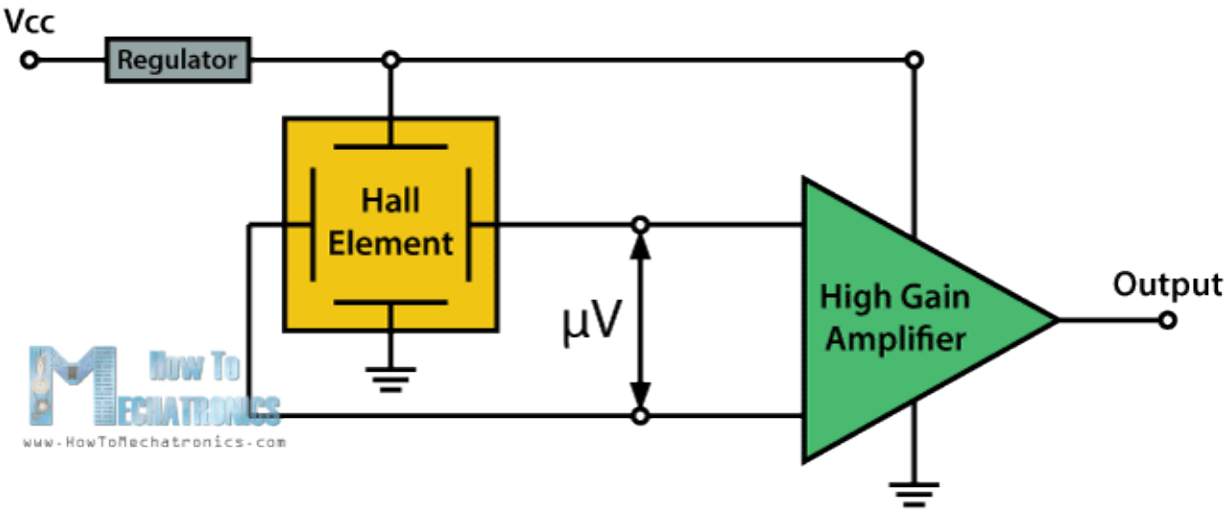

Figure 4. Hall effect sensor operation

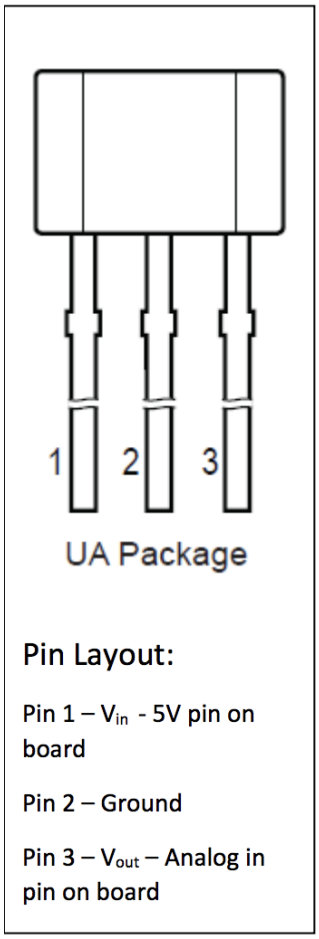

Figure 5. Hall effect sensor integration into circuit

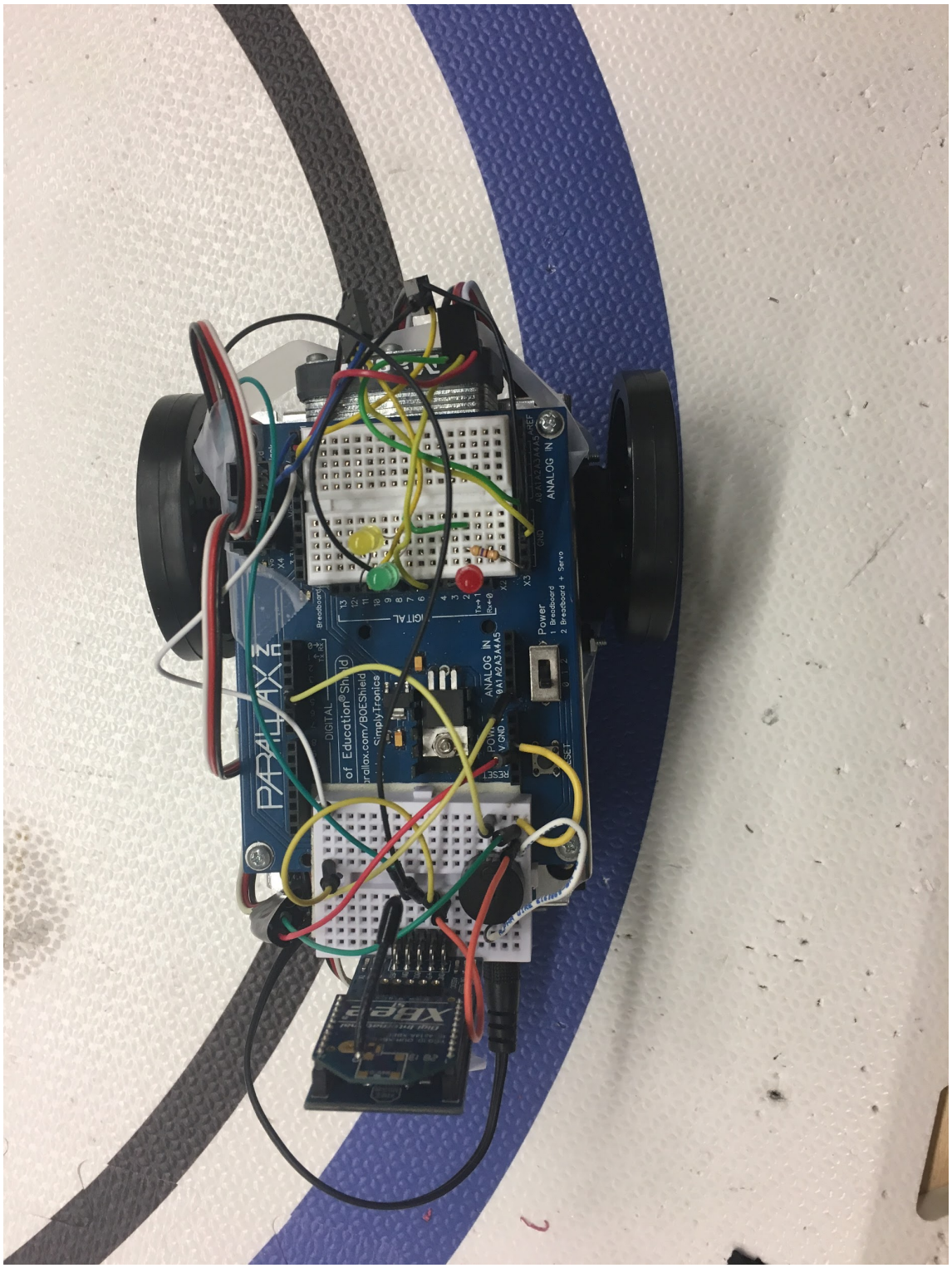

Figure 6. Final bot (Hall-effect sensor hidden by XBee module and QTI sensors hidden by bot)

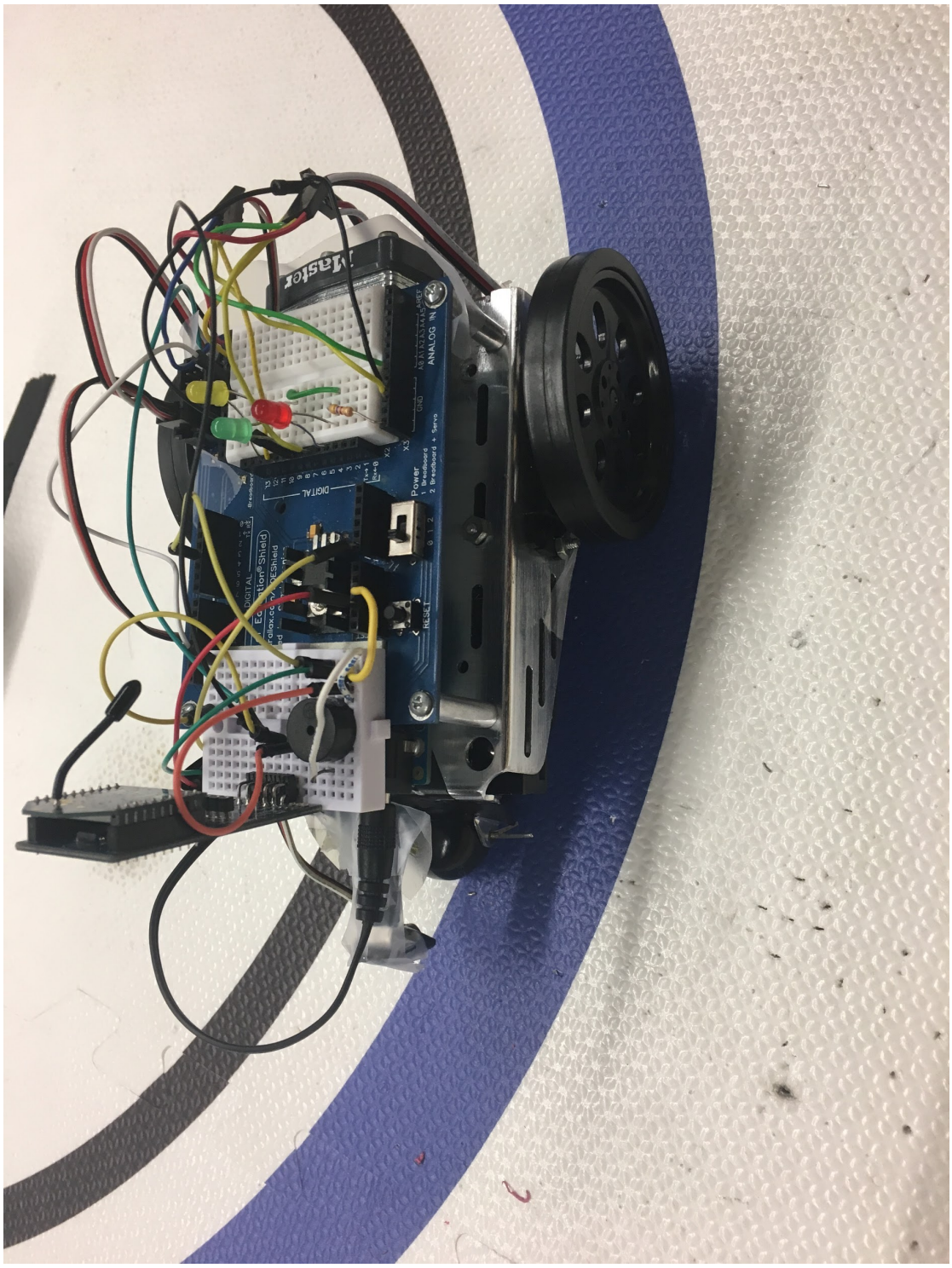

Figure 7. Final bot, Hall-effect sensor wrapped by clear tape and QTI sensors hidden by bot.

Final Code:

```
#include<Servo.h>

Servo servoLeft;  //attaching servos
Servo servoRight;
int R = 5; //right QTI
int M = 7; //middle QTI
int L = 10; //left QTI


int tL = 0; //setting up left time constant
int tM = 0; //setting up middle time constant
int tR = 0; //setting up right time constant
int t = 400; //qti cutoff
int cond = 0; //set up condition variable for loop
int x2 = 0; //our variable
int c2 = 1; //our condition value
int x1 = 0; //set up variables to receive values from other groups
int x3 = 0;
int x4 = 0;
int x5 = 0;
int c1 = 0; //condition variables to determine when a message has been received from another group
int c3 = 0;
int c4 = 0;
int c5 = 0;
const int interval = 500; //interval variable used to stagger message
unsigned long previousMillis = 0; //variable used to measure interval
int ledState = LOW; //LED pin 11 value
int total = 0; //sets up sum and remainder value
//define frequencies for tone
const int c = 261;
const int d = 294;
const int dSH = 622;
const int dS = dSH / 2;
const int e = 329;
const int f = 349;
const int fSH = 740;
```

```
const int fS = fSH / 2;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 493;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int eH = 659;
const int fH = 698;
const int gH = 784;
const int gSH = 830;
const int aH = 880;
const int bH = 987;

const int buzzerPin = 4; //sets up pin connected to speaker
//defining length of notes for song
const int q = 400;
const int qh = q * 1.5;
const int ha = 2 * q;
const int o = 4 * q;
const int ei = q / 2;
char outgoing = ' ';


//defining pins for xbee
#define Rx 17
#define Tx 16
void setup() {
  Serial.begin(9600); //sets up serial monitor(used to troubleshoot)


  servoLeft.attach(12);                      // Attach left signal to pin 13
  servoRight.attach(13);                    // Attach right signal to pin 12
```

```
  delay(500);
  Serial2.begin(9600); //sets up serial for xbee
  delay(500); //delays slow down bot to troubleshoot easier
  pinMode(buzzerPin, OUTPUT); //sets up buzzerpin to write to spaker


}
//This function measures the rc time of the qti sensor to determine whether it is over dark or light ground
long rcTime(int pin) {
  pinMode(pin, OUTPUT); //sets pin to output
  digitalWrite(pin, HIGH); //high voltage to charge capacitor
  delayMicroseconds(230);
  pinMode(pin, INPUT); //switches pin to input
  digitalWrite(pin, LOW);
  long time = micros();
  while
  (digitalRead(pin));
  time = micros() - time; //measures time that has passed
  return time; //returns time when capacitor discharges
}

//Beep Function simplifies tone command and adds delay to properly play notes as a song
void beep(int note, int duration)

{
  //Plays tone on buzzerPin
  tone(buzzerPin, note, duration);
  delay(duration + 10);



}
```

```
//main loop where everything happens
void loop() {
  //setting up output pins
  pinMode(3, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(11, OUTPUT);

  //measuring rcTime of each qti, storing each in a variable:
  tL = rcTime(L);
  tM = rcTime(M);
  tR = rcTime(R);

  //Serial prints for troubleshooting:
  //Serial.println("Left");
  //Serial.println(tL);
  //Serial.println("Middle");
  //Serial.println(tM);
  //Serial.println("Right");
  //Serial.println(tR);
  //Serial.println(cond);

  //Main line following code:
  if ((tL < t) && (tM > t) && (tR < t) && (cond == 0)) {
    servoLeft.writeMicroseconds(1550);
    servoRight.writeMicroseconds(1450);
    //straight, only center sees black and outward two see white
  }



  else if ((tL < t) && (tM < t) && (tR > t) && (cond == 0)) {
    servoLeft.writeMicroseconds(1580);
    servoRight.writeMicroseconds(1510);
    //right turn, right sensor sees black and other two see white
  }

  else if ((tL > t) && (tM < t) && (tR < t) && (cond == 0)) {
    servoLeft.writeMicroseconds(1490);
    servoRight.writeMicroseconds(1420);
    //left turn, left sensor sees black
  }
```

```
else if ((tL > t) && (tM > t) && (tR > t) && (cond == 0)) {
   //all three see black
   cond = 1; //changes condition so bot can run through other sections of code
   servoLeft.writeMicroseconds(1500);
   servoRight.writeMicroseconds(1500);
   //bot stops
   delay(500);
   servoLeft.writeMicroseconds(1520);
   servoRight.writeMicroseconds(1480);
   //bot inches forward slightly
   delay(400);
   servoLeft.writeMicroseconds(1500);
   servoRight.writeMicroseconds(1500);
   //bot stops
   delay(500);
   servoLeft.writeMicroseconds(1450);
   servoRight.writeMicroseconds(1450);
   //90 degree turn
   delay(1000);
   servoLeft.writeMicroseconds(1470);
   servoRight.writeMicroseconds(1530);
   //reverse towards target (due to placement of sensor on back of bot)
   delay(1000);
   servoLeft.writeMicroseconds(1500);
   servoRight.writeMicroseconds(1500);
   //stop
   delay(1000);
 }
```

```
  float reading = (analogRead(0));
  //Output of Hall Effect Sensor (integer between 0-1024)

  float mag = reading * 5.0;
  // Converts reading value to volts Signal


  mag /= 1024.0;
  //Divides by 1024 giving the sensor output signal in volts

  mag = mag - 2.50;
  //Subtacts the zero field output of the sensor to provide pole sensing

  mag = mag * 5000;
  //Sets units of output to Gauss (5mV/Gauss)

  //Serial.print("mag strength = "); // Display sensor value(troubleshooting)
  Serial.println(mag, 4);
```

```
//Main sensing:
if (cond == 1) {

  unsigned long currentMillis = millis(); //measures current time
  if (currentMillis - previousMillis >= interval) { //after interval time runs through sending code
    previousMillis = currentMillis; //resets counter

    if (abs(mag) > 1600) { //sensing black
      digitalWrite(11, HIGH); //turns on LED to signal black
      outgoing = '6'; //sets up 6 as the outgoing variable
      Serial2.print(outgoing); //sends 6 as a character to other groups
      x2 = 2; //sets our variable to 2(gold)


    }
    else if ((abs(mag) > 180) && (abs(mag) < 1600)) { //sensing gray
      if (ledState == LOW) { //flips pin 11 on and off
        ledState = HIGH;
      }
      else {
        ledState = LOW;
      }
      digitalWrite(11, ledState);
      outgoing = '5'; //sets up 5 as the outgoing variable
      Serial2.print(outgoing); //sends 5 as a character to other groups
      x2 = 1; //sets our variable to 1(silver)
    }
    else { //sensing white
      digitalWrite(11, LOW); //no LED light
      outgoing = '4'; //sets up 4 as the outgoing variable
      Serial2.print(outgoing); //sends 4 as a character to other groups
      x2 = 0;  //sets our variable to 0(bronze)


    }
  }
```

```
if (Serial2.available()) {
  char incoming = Serial2.read(); //sets incoming variable as that read by XBee

  //Serial.println(incoming); (troubleshooting)

  //This section converts the incoming character to an integer:
  //Each if statement determines which range the message came from (1-3, a-c,d-f,g-i)
  //whats it finds the correct location, it assigns the x variable to a 0, 1, or 2 based on the message received
  //it flips the c variable to a 1 so that it stops listening for that groups message
  //it plays a tone to let us know it has received a message

  if ((incoming - 48 < 4) && (incoming - 48 > 0) && (c1 == 0)) {
    x1 = incoming - 49;
    c1 = 1;
    tone(4, 220, 200);
    Serial.println(x1);
    Serial.println(c1);
  }
  else if ((incoming - 96 < 4) && (incoming - 96 > 0) && (c3 == 0)) {
    x3 = incoming - 97;
    c3 = 1;
    Serial.println(x3);
    tone(4, 220, 200);
    //Serial.println(c3);
  }
  else if ((incoming - 96 < 7) && (incoming - 96 > 3)  && (c4 == 0)) {
    x4 = incoming - 100;
    c4 = 1;
    Serial.println(x4);
    tone(4, 220, 200);
    //Serial.println(c4);
  }
```

```
      else if ((incoming - 96 < 10) && (incoming - 96 > 6)  && (c5 == 0)) {
        x5 = incoming - 103;
        c5 = 1;
        Serial.println(x5);
        tone(4, 220, 200);
        //Serial.println(c5);
      }
    }
  }
```

```
//Once all c variables equal 1, the bot has received a message from each group and can calculate the remainder

if ((c1 == 1)  && (c2 == 1)  && (c3 == 1)  && (c4 == 1)  && (c5 == 1) && (cond == 1)) {
  cond = 2; //changes to final demonstration condition
  total = (x1 + x2 + x3 + x4 + x5) % 3; //calculates remainder
  outgoing = outgoing; //helps get proper value into this part of code

  // Serial.print("Total: ");
  // Serial.println(total);
}
if (cond == 2) {


  digitalWrite(6, LOW); //writes all LEDs to low
  digitalWrite(3, LOW);
  digitalWrite(11, LOW);

  if (total == 0) { //bronze metal, bot dances back and forth and ccontinues sending message

    digitalWrite(6, HIGH); //displays green light
    servoLeft.writeMicroseconds(1550);
    servoRight.writeMicroseconds(1450);
    Serial2.print(outgoing);
    delay(1000);
    servoLeft.writeMicroseconds(1450);
    servoRight.writeMicroseconds(1550);
    delay(1000);
    Serial2.print(outgoing);

  }
```

```
  else if (total == 1) {
    //plays light show by flashing green and red LEDs, and continues sending message
    digitalWrite(3, HIGH);
    digitalWrite(6, LOW);
    Serial2.print(outgoing);
    delay(500);
    digitalWrite(6, HIGH);
    digitalWrite(3, LOW);
    delay(500);
    Serial2.print(outgoing);



  }
```

```
else if (total == 2) {
  //turns on both lights, plays song, sends message
  digitalWrite(3, HIGH);
  digitalWrite(6, HIGH);
  beep(b, q);
  beep(cSH, qh);
  Serial2.print(outgoing);
  beep(cSH, qh);
  beep(fSH, q);
  beep(eH, qh);
  Serial2.print(outgoing);
  beep(dH, qh);
  beep(cSH, q);
  beep(b, qh);
  beep(b, qh);
  beep(cSH, q);
  beep(a, ha * 1.5);
  Serial2.print(outgoing);
  beep(b, q);
  beep(cSH, qh);
  beep(cSH, qh);
  beep(fSH, q);
  beep(eH, qh);
  beep(cSH, qh);
  beep(eH, q);
  beep(b, o);
  Serial2.print(outgoing);
  delay(q * 3);
```

```
  beep(gSH, q);
  beep(aH, qh);
  beep(gSH, qh);
  beep(fSH, q);
  beep(eH, qh);
  beep(dH, qh);
  beep(cSH, q);
  beep(b, qh);
  Serial2.print(outgoing);
  beep(eH, qh);
  beep(dH, q);
  beep(cSH, ha * 1.5);
  beep(gSH, q);
  Serial2.print(outgoing);
  beep(aH, qh);
  beep(gSH, qh);
  Serial2.print(outgoing);
  beep(fSH, q);
  beep(eH, qh);
  beep(aH, qh);
  beep(bH, qh);
  beep(bH, o + q);
  delay(ei);
  beep(aH, ei);
  beep(aH, ei);
  Serial2.print(outgoing);
  beep(gSH, ei);
  beep(gSH, ei);
  beep(eH, ei);
  beep(fSH, qh);
  beep(fSH, ei);
  beep(gSH, q);
  Serial2.print(outgoing);
```

```
    beep(gSH, ei);
    beep(gSH, q);
    beep(aH, ei);
    Serial2.print(outgoing);
    beep(aH, q);
    beep(aH, ei);
    Serial2.print(outgoing);
    beep(gSH, ei);
    beep(gSH, ei);
    beep(eH, ei);
    beep(fSH, q);
    beep(aH, ei);
    beep(bH, q);
    Serial2.print(outgoing);
    beep(aH, ei);
    beep(gSH, ei);
    beep(gSH, q);
    beep(fSH, qh);
    beep(aH, ei);
    Serial2.print(outgoing);
    beep(gSH, ei);
    beep(gSH, ei);
    beep(eH, ei);
    beep(fSH, q);
    beep(aH, q);
    beep(eH, q);
    Serial2.print(outgoing);
    delay(ei);
    beep(aH, ei);
    beep(gSH, ei);
    beep(fSH, ei);
    Serial2.print(outgoing);
    beep(gSH, ei);
    beep(aH, ha + ei);
    beep(aH, q);
    beep(eH, ei);
```

```
beep(aH, ei);
beep(aH, ei);
beep(gSH, ei);
beep(gSH, ei);
beep(eH, ei);
beep(fSH, qh);
beep(fSH, ei);
beep(gSH, q);
beep(gSH, ei);
beep(gSH, q);
beep(aH, ei);
Serial2.print(outgoing);
beep(aH, q);
beep(aH, ei);
beep(gSH, ei);
beep(gSH, ei);
Serial2.print(outgoing);
beep(eH, ei);
beep(fSH, q);
beep(aH, ei);
beep(bH, q);
Serial2.print(outgoing);
beep(aH, ei);
beep(gSH, ei);
beep(gSH, q);
beep(fSH, qh);
Serial2.print(outgoing);
beep(aH, ei);
beep(gSH, ei);
beep(gSH, ei);
beep(eH, ei);
beep(fSH, q);
beep(aH, q);
beep(eH, q);
delay(ei);
```

```
      Serial2.print(outgoing);
      beep(aH, ei);
      beep(gSH, ei);
      Serial2.print(outgoing);
      beep(fSH, ei);
      beep(gSH, ei);
      beep(aH, ha + ei);
      beep(aH, q);
      beep(eH, ei);
      beep(eH, q);
      beep(cSH, ei);
      beep(eH, ei);
      beep(b, qh);
      delay(3 * q);
    }
  }
}
```

VII. Citation

What is Hall Effect and How Hall Effect Sensors Work. Dejan Nedelkovski (2017)

https://howtomechatronics.com/how-it-works/electrical-engineering/hall-effect-hall-effect-sensors-work/

QTI Line Follower AppKit for the BOE-Bot.

https://www.parallax.com/sites/default/files/downloads/28108-QTI-Line-Follower-Guide-v2.1.pdf