Final Design Report

IDC Group 4: Michael Kim, Juan Philippe, and David Tran
TA: Patrick Grady
ECE 110L-01L

Date Submitted: 27 April 2016

We have adhered to the Duke Community Standard in completing this assignment.

Contents

1 Abstract

The integrated design challenge was created to apply and enhance concepts learned in lecture and in early labs during the ECE110 curriculum, while also placing an emphasis on small group work and inter-group cooperation. Broadly speaking, group goals were to design and build a robot that would perform an individual task, record a result, communicate with the other robots, and perform an action with the rest of the robots dependent on the results of the individual tasks. The challenge was a success in that this bot was able to perform the tasks, measure the correct values, and communicate effectively with the other robots. It was also a success in that the experience greatly enhanced engineering-related concepts and skills such as organization, code modification, circuit design, and group coordination and communication.

2 Introduction

2.1 Identification of Problem

The integrated design challenge and the related problems can be broken down into the team challenge, in which all groups participate and perform the same actions, and the group challenge, in which a particular bot judges a particular event.

2.1.1 Team Challenge

The primary objective of the Integrated Design Challenge was the holistic success of five robots in not only individual challenges that required the cooperation of multiple sensors within the individual bot, but also on group coordination based on inter-robot communication. More precisely, while each team designed and implemented a robot that was individually and particularly suited to being successful in judging its assigned sport, each robot was also capable of communicating its particular results to and with the others and making decisions on that basis.

To be specific, after judging the assigned event, each robot transmitted a 0, 1, or 2 (equating to a bronze, silver, or gold medal respectively). The cumulative score was then determined by a master bot and all the bots engaged in a coordinated activity: a dance, a light show, or playing of an "anthem" based on the cumulative score.

2.1.2 Group Challenge

Group 4 was tasked with judging the basketball event. To successfully judge this event, the bot had to parade the event space by following the black circle circumscribing the court once fully. Then, the bot had to find the basketball (an RFID tag) and determine the location of the final shot. The final shot could have taken place from one of three possible locations that were called: 3-point, 2-point, and free throw. Initially this was believed to be a rangefinding task. However, this was later determined not to be the case. This realization will be further discussed in Section 3. The 3-point shot would garner a gold medal, the 2-point a silver, and the free throw a bronze. Once this determination was made, the bot would light up one of three colored LEDs to communicate the medal and communicate this conclusion with the other bots by XBEE. Finally, the bot would wait for the master bot to communicate which final team activity was appropriate.

2.2 Importance of Problem

While this particular situation has been contrived as a learning exercise, the IDC was designed to be a practical application of the theoretical concepts learned previously in lab and in lecture. There are two skills primarily exercised: hardware (i.e. construction of circuits) and software (i.e. writing the relevant code). Construction of the actual circuits reinforced basic concepts such as Ohm's law, the reading of schematics, and proper circuitry (i.e. avoiding shorts). The software side of the project required a basic understanding of coding fundamentals and the ability to modify provided Arduino code. In aggregate, the IDC was particularly effective because it required not only a foundational knowledge of both the software and hardware skills listed, but also required the group to extrapolate and infer based on that knowledge to trouble shoot and design a successful BOE-bot.

2.3 Proposal of Solution

There were three sub-tasks required of the BOE-bot to be successful: line following, determination of the final shot, and communication (and the team activity).

2.3.1 Line Following

The BOE-bot was required to parade around the event space once by following the black ring. Additionally, because of the nature of judging the basketball event, it was critical that our bot stopped exactly on the hash mark and make a crisp 90° turn.

To that end, four QTI sensors were used. The inner two, placed just a little further apart than the width of the black line, were primarily used for line following. The bot would sweep back and forth. If the left QTI sensed black, the bot would know that it was turned too far to the right (since the left sensor is over the black line). It would then correct by sweeping slightly to the left. No correction would be made if the left sensor sensed white. The same logic held for the right sensor. If the right QTI sensed black, the bot would know that it was turned too far to the left and correct by sweeping to the right. In this manner, the bot would making small sweeping corrections trying to keep both the left and right sensors over the white portion of the ground just to the left and right of the black line.

The outer two QTI's were critical for the detection of and proper alignment with the hash mark. The outer QTI's were placed wide enough where they always sensed white except at the hash mark. Therefore, when all four QTI's sensed black, the bot would perfectly aligned with the hash mark at times (see Section 4.2 for line-following improvements).

The bot was started from the hash mark for simplicity and circumnavigated the circle once until the QTI's determined that the bot had returned to the hash. Once in the starting position, the bot was hard coded to make a right angle left turn so that it would be directly in line with the backboard and ready to begin the next phase of the challenge.

2.3.2 Determination of the Final Shot

There were three possible positions at which the RFID tag could have been placed. All were in line with the hash mark and the backboard. The furthest from the backboard was just above the three point arc and denoted a three point shot. The next furthest from the

backboard was at the free throw line and denoted a two point shot. The closest to the backboard was just in front of the backboard and denoted a one point shot. Because there were only three possible positions, there was no need for any the BOE-bot to have any range-finding capability.

To clarify, if the challenge allowed for the ball to be placed anywhere on the straight line between the hash mark and the backboard, the BOE-bot would have had to been coded with the ranges that corresponded to a type of shot (e.g. any shot between the hash mark and the three point arc would have been a three point shot) and range-finding capability would have been required to determine the distances that corresponded to those ranges.

In this case, the BOE-bot simply had to check at three positions to determine the final shot's location. After turning at the hash mark, the bot would proceed forward to the location of the three point shot and stop. The RFID sensor would check for an RFID tag. If the tag was sensed, the bot could stop and communicate that the final shot was a three point. If no tag was sensed, the bot would continue to the position of the two point shot and stop. If the tag was sensed, again, the bot could stop and communicate that the final shot was a two point. If no tag was sensed, the bot would continue to the position of the one point shot and stop and sense for an RFID tag again.

2.3.3 Communication and Team Activity

The bot used a XBEE module to communicate this group's judging of the basketball event. Instead of each group sending a 0, 1, or 2, each group had three letters assigned to it and those three letters were associated with one of the three possible values. This way, it would be easier to trouble shoot as it could be easily determined which group failed to transmit in such a case. The master bot then calculated a cumulative score based on the letters it received.

If the modulus three sum results in a team gold medal, the master bot would transmit a letter that caused all the bots to play the anthem. If the modulus three sum results in a team silver medal, the master bot would transmit a letter that caused all the bots to put on a light show. If the modulus three sum results in a team bronze medal, the master bot would transmit a letter that caused all the bots to dance.

This group provided a dance code to the entire team. A light show code was provided by another group. And the anthem was a pre-made code that played the Super Mario theme that was adapted by another group.

3 Experimental Procedures and Results

3.1 Week 1: 1 March 2016 - Introduction

Week 1 was largely a preliminary stage during which the group planned the entire IDC. A problem statement, which is presented in a final, comprehensive form in the previous Section 2.1, was developed along with a statement regarding our objectives and deliverables. These are presented in final form in Section 2.3. Additionally, as part of the planning, a Gantt chart with task assignments was made. This chart would be used as part of each weeks'

progress reports for the duration of the project. After conducting a trade study to determine the best sensor with which to conduct the range finding, a cost estimate of the BOE-bot as provided was conducted. The cost estimate is presented in final form in Section 4.1. All the cited sections are simply retrospective and final forms of the corresponding sections first included in Week 1's Conceptual Design Report [1].

3.2 Week 2: 8 March 2016 - Conceptual Design, Communication

Week 2 of the IDC revolved around incorporating the XBee module into our bot in order to complete a Communication demo. Successful completion of this demo meant that our bot would be capable of communicating with the others for the Team Challenge, though the demo was a more simplistic task than the final communication challenge. We installed the XBee module as instructed by the XBee manual, then implemented a simple push button circuit, shown in Figure 6. We wrote code such that when the button was pushed, the XBee module would send out a letter. This was simple: when the button was pushed, the voltage across pin 8 would go 'HIGH' because the pushed button would close the circuit. The code had an if statement; if pin 8 was 'HIGH' then the bot would send out a character through the XBee to be received by the other bots. Furthermore, we installed an LED connected to an outgoing pin- pin 3 at the time. We then modified our code, using another if statement. If the bot received the character 'P' from another bot, then it would set pin 3 to 'HIGH,' applying a voltage across the LED circuit, causing the LED to illuminate momentarily. This task was early on, so the bot was still relatively simple and easy to work with, resulting in few mistakes and no real problems in completing the demo. As soon as we correctly installed the components and ran our code we got the XBee and the bot working up to par.

3.3 Week 3: 29 March 2016 - Line Following, Sensing, Communication

To prepare for Week 3 of the IDC, our bot needed to be able to line follow, detect an RFID tag, and light an LED based on the tag's distance from the backboard. Therefore, the major sensors we knew we needed were two QTI sensors, one RFID sensor, and one PING sensor. For our line following, we wrote code comprised of a truth table for our two QTI sensors. Each of the sensors can either receive infrared light reflectivity values above or below a threshold (0 or 1, respectively). This threshold value was determined by wiring the QTI and printing its detected values for the black line and the white surrounding to the serial monitor. For example, the white surface might have values of 120 while the black line has 240. To make a threshold for this example, we would have chosen the average of the two, 180. This would ensure a distinct difference in values with which the bot would adjust its movement (see Section 2.3.1 for more information on our line following approach). We saw high success rate using truth table approach when testing on the different line patterns. The only line pattern that we sometimes had issues with was the square since its corners were very sharp. One way we combated this was by making one of the bot's wheels turn backwards while the other turned forwards whenever the bot actually needed to turn. This prevented both QTI sensors from passing completely outside of the square while the bot was still turning. The QTI sensors sometimes passed completely across due to the geometry of the BOE-Bot itself and the configuration of the servos and the spherical wheel.

For the sensing portion of the challenge, we mounted the RFID sensor in front of the

bot, parallel to the ground so that the sensor could easily detect the tag on the ground (see Figure 4 to see our RFID mount configuration). We mounted three LEDs on the breadboard to display which of the three shot types was sensed by our bot (see Figure 7 for schematic). Green is binary 10 (three point shot), yellow is binary 01 (two point shot), and red is binary 0 (1 point shot).

To determine the type of shot, we implemented the PING sensor that calculated the distance between the sensor and the backboard. This integer distance value was calculated using code from the PING template code from learn.parallax.com. We then hard-coded distance values in "if" statements so that when the RFID sensor detected a tag, the PING sensor would calculate the distance and the bot would determine which "if" statement to enter. Once inside an if statement, the respective LED was set to "HIGH". This approach of coding seemed most logically and straightforward to code it utilizes the specific sequence of detecting then pinging for distance. The only thing that we need to cautious of was to make sure that the PING sensor was pinging against the backboard and not the wooden frame surrounding the IDC arena. As a result, for the trials, the bot needed to be directly facing the backboard. More information on line following and turning at the hash will be presented in the next section.

3.4 Week 4: 5 April 2016 - Integrated Sensing

For this week, our bot was expected to circumnavigate, turn towards the backboard, increment forward until it detects the RFID tag, and finally display the shot type using LEDs. The bot's line following was accurate up until it reached the hash. Since the bot would move forward, turn for about 0.5 seconds, and move forward again in order to line follow, sometimes it would turn at the last instant before stopping at the hash. Sometimes, it would align perfectly perpendicular to the hash. This meant that we could not hard code a 90° turn since the bot would start a slightly different position. Therefore, we implemented the PING sensor to allow the bot to turn until the PING sensor detects the backboard. The distance between the hash and backboard was about 16 inches, the bot would turn and send pings at the same time until a distance greater than 14 inches was detected. Since there were no closer objects within the area of turning, we could have simply sent signals throughout the entire turn. However, to be extra cautious, we used a hard-coded delay so that the bot would send pings just before it faced the backboard. Some issues we observed from this approach was that the PING sensor would detect the backboard "early" in that the bot would stop slightly before it was directly facing the backboard. To temporarily compensate for this, we set a delay so that after the backboard was detected, the bot would turn slightly more. In our trials, we observed a much higher success rate because the delay helped the RFID sensor get within its sensing range of the tag at the 1-point shot.

After a distance of greater than 14 inches was calculated, the bot would stop after a short delay and start to increment forward every two inches and check for the tag in between increments. The reason we did not program the bot to move forward while continuously checking for the sensor was because after the tag was detected, the servos would twitch excessively. This may be due to a conflict in SoftwareSerial ports or a specific conflict of using these sensors simultaneously. This is why we detached the servos when the bot stopped in between each increment and then we reattached them before the bot started moving forward again. This substantially decreased the servo twitching. Once the tag was

detected, the PING sensor initiated its sensing and the respective LED was lit based on the calculated distance, exactly like the week before. Overall, the bot had an adequate success rate given the small amount of time we had to develop the combination of circumnavigation and sensing.

3.5 Week 5: 12 April 2016 - Team Sensing

For the Team Sensing demo week, our group was assigned the responsibility of writing the dance code used for the team bronze medal (see lines 239-260 in Figure 1). Our design considerations was to use servos in a repetitive way that makes the bot's movements look predetermined. Forward and backward movements and swivels were the simplest to code and were the most effective for a dance. To create repetitive movements, we used while loops almost as for loops and implemented servo commands within those loops. Also, we created Xbee transmitting code for the communication of letters with the master bot (see lines 146-164 in Figure 1). Our approach was to re-initialize the Xbee port to guarantee sending and receiving. To ensure that the bot keeps sending a J, K, or L until it receives an X, Y, or Z, we created a while loop that checks for the X, Y, or Z and subsequently sends our letter once and loops back again. This way, it is guaranteed that our bot keeps sending until it receives the correct letter. In our team trials and meetings, our bot was able to communicate successfully with the master bot.

Some hardware changes made were the addition of the piezo-speaker and an extra white breadboard. The original Arduino breadboard was very crowded, and given the need for a piezo-speaker, a new breadboard was greatly needed. The breadboard was placed on top towards the back (on the side of the black spherical wheel) where voltage and ground ports were available. Fortunately, there was one digital pin open to which we connected the speaker (see Figure 5). We also moved our three LEDs to the white breadboard for convenient observations. After the wiring, the LEDs and speaker always worked as expected.

3.6 Week 6: 19 April 2016 - Full System and Oral Defense

The major design consideration going into the final week of development was to implement a method of completing the challenge without using the PING sensor. In the Team Sensing demo from the week before, our group was the slowest and least accurate in completing individual group phases. At times our PING sensor would detect the backboard too early and thus stop well before completing a 90° CCW turn. This occurred about 33% of the time on the day of the test, which is unacceptable. Therefore, we went back to the drawing board, re-read our specific assignment on the IDC handout. Then, it came to our minds that since the tag will be in only three distinct locations, all aligned between the hash and the backboard, why not simply check for each location in a sequence? There would be no need for a PING sensor since we know the sequence of shot types. Thus, we simply made a while loop where the bot would move toward the backboard in three increments, checking for the tag between each (see line 108 in Figure 1). After each increment, an integer variable would increase by 1 and then break out of the while loop after the tag was detected by the RFID sensor. Then we created if statements based on the value of our variable to light the respective LED and send the corresponding letter to the master bot. Using this method, our bot would not have to rely on the backboard at all but simply (and quickly) check for the tag at only three distinct locations.

However, the issue that arose from this seemingly effective new approach was that we also had to implement a new turning mechanism for our bot. Since the PING sensor was unreliable for our turning, the only other method we could think of was to hard code the 90° turn. Therefore, the success of turning relied heavily on how well the bot perpendicularly aligned with the hash. To ensure an almost perfect alignment, we added two outer QTI sensors solely for the hash alignment and moved the two inner QTI sensors closely for a smoother, more precise line follow (see line 87 and 88 in Figure 1 for outer-QTI code). In addition, we eliminated the delay on infrared sensing for the QTI sensors so that the bot would make many adjustments to smoothly line follow. The culmination of these changes prevented the bot from make last second large turns before stopping at the hash that caused misalignment, which is why we avoided this hard-coding method during the first couple weeks of the IDC. Although there was some error in alignment, the bot was still able to detect even the furthest tag after turning and incrementing. After performing several trials, sometimes even with the tag an inch away from its actual spot, we observed that the bot had a 95% chance of accurately sensing and navigating.

4 Analysis and Discussion

4.1 Statement of Cost

The initial BOE-Bot as provided included two lithium ion batteries, a 2 A fuse and power pack, a 7.5 V power supply, a XBEE Module and adapter, two continuous servo motors, a USB port, an ATmega2560 board, the BOE Shield board, two wheels, and a chassis. This made the starting cost about $300.

As part of the primary design, four QTI sensors ($9.99 each), a RFID reader module ($44.99), a PING))) ultrasonic sensor ($29.99), and about $5 of miscellaneous mounting hardware and wires were added. This was an additional cost of $120 bringing the total to $420.

Finally, a peripheral breadboard ($3.50) and a piezospeaker ($2) were added to complete the construction. This was an additional cost of $5.50 bringing the total to $425.50.

The initial $300 cost of the BOE-bot cannot be changed. And as seen in the cost breakdown, the sensors are the biggest additional costs. Therefore it is by using only the minimum sensors in the most efficient manner that cost can be minizmized. However, this use of the minimum sensors must also be weighed against the added benefit of additional sensors for better sensing and better function. Tangentially, while there could be small savings made by more efficient wiring and construction (i.e. through using less wires and hardware) this savings is not particularly useful as this would be on the order of cents and not significant magnitude - since the bot already costs hundreds of dollars.

The final design omitted the PING))) sensor after it was deemed unnecessary based on a deeper understanding of the known values in our problem as discussed in previous sections. Forgoing this sensor would have been a savings of about $30. Additionally, given more time, the line following could have been fine tuned so that only two QTI sensors would have been necessary. This would have been a savings of about $20. However, the design is

definitely optimized by the use of four QTI sensors as the outer two QTI's significantly increase the precision with which the BOE-bot can align with the hash mark. This alignment was critical to the performance of our bot and therefore it is difficult to conclude that the $20 savings through the removal of two QTI's would have been worthwhile.

4.2 Analysis of Final Design

Please see Sections 3.1-7 for a review of design justifications in their entirety from each week. One issue that fortunately did not negatively impact our bot's performance was the slightly incomplete 90° CCW turn at the hash. This slight error in turning was expected due to the bot's imperfect line-up with the hash. However, it was deemed insignificant to the bot's performance since the RFID tag was able to sense the tag within a few inches. Still, one way that we could have improved the turning, if given more time, would be to reprogram the QTI line-following by having two QTI sensors flush in the exactly between the two wheels along with two outer QTI sensors. This way, the bot would move forward when both inner QTI sensors receive low infrared light (i.e. detect the line since it was fairly wide) and make respective adjustments using coded truth table. This would ensure a much smoother line-follow while also guaranteeing a perpendicular line-up with the hash using the outer QTI sensors. Another minor improvement would be to replace the rubber bands on the wheels since they may have worn down and acquired "low-friction" patches.

Some other key design aspects was to have the RFID sensor in front of the bot in order to sense the tag as quickly as possible and also to avoid running over the backboard for the "bronze" shot. Also, the extra white breadboard was placed toward the back where extra voltage and ground ports were available. Other than the line-following, there were no other persistent issues that we encountered given that our bot's team communication and RFID tag sensing worked perfectly in every trial of the IDC.

4.3 Proposals for Design Changes and Improvements

Please see Section 2.3.2 for a review of the logic used to determine the location of the final shot. In hindsight, if the RFID tag was not sensed at the three point or two point positions, the bot did not have to proceed to the one point position to determine where the tag was as no tag at the three or two point positions immediately requires that the tag be at the one point position. Therefore, the logic could have been streamlined such that no tag being detected at the three and two point positions allowed the bot to communicate that the final shot was a one point shot immediately. This would have shortened the time required for the bot to judge the event.

One of the biggest improvements made during the process was to unplug all the circuits and completely rewire the bot in an organized fashion. In the future, the biggest "improvement" would be to organize the circuitry for the bot all together and build based on that, rather than adding elements haphazardly over time. The messy robot caused many problems because it made it very easy to make a mistake and very hard to find where it occurred. This resulted in short circuits, misuses of pins, infinite loops and a lot of wasted time because the bot was complicated to work with and hard to troubleshoot. All of this went away when the bot was rewired. Clarity and organization are key.

5 Conclusion

The IDC is fundamentally a learning challenge, meaning that the primary, task-based objectives exist alongside secondary objectives based on learning, applying existing knowledge, and coordinating and communicating with others. The engineering objectives of this challenge were met successfully, as the bot completed all tasks correctly, accurately recorded scores, and communicated with the other bots flawlessly. Working to fulfill these goals led us to complete our secondary objectives. We learned about circuit design while building the various circuits needed to make the bot do its job; we learned to write and modify code in Arduino for a range of different tasks; we applied our knowledge on signals to work with sensors; and most importantly, we worked together fluidly and efficiently to accomplish our tasks.

One important thing we learned was to be flexible in our design approaches. For a while we felt stuck because we could not improve the score-identifying accuracy of our bot; slight errors in positioning resulted in the PING sensor frequently giving poor distance measurements. As soon as we opened our mind to another approach, we found a solution that completely removed our problem, potentially saving us hours of work trying to fix our first method.

Additionally, we learned how imperative it is to work on an organized platform. The three members of this group worked very well together. We communicated frequently, delegated work appropriately, and committed consistently to working on the bot every week. Despite all this, we found ourselves falling behind by the fourth week because our bot was so error prone- a result of our hardware. The bot's wiring was so tangled, cramped and disorganized that at any moment something could come loose, and we would have to spend 15 minutes finding the tiny, unplugged wire and another 5 minutes plugging it back in to the correct pin through the mess. We eventually realized how much time and effort this cost us, so we decided to pull apart and rewire the bot in a clean, organized fashion. The effect of the rebuild was big, making our lives much easier. Whether we were adding a new component or troubleshooting, the amount of time we spent getting our hardware to work dropped quickly, as it became much easier to work with individual components and to find sources of trouble. The IDC was ultimately a success, helping us learn what it means to be an engineer, challenging us and forcing us to cooperate effectively.

6 Appendices

The following appendices include the complete Arduino code, pictures of the BOE-bot, and relevant schematics.

```
#include <Servo.h>
#include <SoftwareSerial.h>
#include "pitches.h"

#define enablePin  51
#define rxPin      53
#define txPin      11
#define BUFSIZE    11
#define RFID_START  0x0A
#define RFID_STOP   0x0D
#define RLED 9
#define GLED 3
#define YLED 2
#define Rx 10 //DOUT
#define Tx 11 //DIN
SoftwareSerial rfidSerial(rxPin, txPin);
SoftwareSerial Xbee (Rx, Tx);

Servo servoLeft;
Servo servoRight;
long RCTime(int sensorIn);
void writeServo(float leftSpeed, float rightSpeed);

const int pingPin = 6;
unsigned int duration, inches;
int song = 0;

void setup() {
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(8,INPUT);
    pinMode(9,OUTPUT);

    Serial.begin(9600);
    Xbee.begin(9600);

    pinMode(enablePin, OUTPUT);
    pinMode(rxPin, INPUT);
    digitalWrite(enablePin, HIGH);
    rfidSerial.begin(2400);
    Serial.flush();

    servoLeft.attach(13);
    servoRight.attach(12);

    LineFollow();
    RFID();
}

void loop() { }

void writeServo(float leftSpeed, float rightSpeed) {
    int left = 1500 + leftSpeed * 500.0;
    int right = 1500 - rightSpeed * 500.0;
    servoLeft.writeMicroseconds(left);
    servoRight.writeMicroseconds(right);
}

long RCTime(int sensorIn) {
    long duration = 0;
    pinMode(sensorIn, OUTPUT);
    digitalWrite(sensorIn, HIGH);
    delay(1);
    pinMode(sensorIn, INPUT);
    digitalWrite(sensorIn, LOW);
    while(digitalRead(sensorIn)) {
        duration++;
    }
    return duration;
}

void LineFollow() {
    while(1) {
        //Serial.println(RCTime(7));        //RCTime(7) LEFT sensor
        Serial.println(RCTime(4));        //RCTime(4) RIGHT sensor
        //Serial.println(RCTime(33));     //outside LEFT sensor
        //Serial.println(RCTime(31));     //outside RIGHT sensor
        //delay(5);
        int threshold = 40;
        int left = RCTime(7) < threshold;
        int right = RCTime(4) < threshold;

        if(right && left)   writeServo(1,1);   //forward
        else if(right)      writeServo(0,1);   //turn right
        else if(left)       writeServo(1,0);   //turn left
        else if(RCTime(33) < threshold) writeServo(0,1);
        else if(RCTime(31) < threshold) writeServo(1,0);
        else {
            writeServo(1,1);
            delay(240);
            writeServo(-1,1);
            delay(650);
            writeServo(0,0);
            delay(100);
            break;
        }
    }
}

void RFID() {
    digitalWrite(enablePin, LOW);
    digitalWrite(3,LOW);
    digitalWrite(2,LOW);
    digitalWrite(9,LOW);
    delay(50);
    int check = 0;
    while(1) {
        servoLeft.attach(13);
        servoRight.attach(12);
        writeServo(1,1);
        if (check == 0) delay(275);
        else if (check == 1) delay(750);
        else if (check == 2) delay(650);
        writeServo(0,0);
        delay(500);
        servoLeft.detach();
        servoRight.detach();
        check++;
        //delay(200);
        if(rfidSerial.available()) {
            if (check == 1) {
                digitalWrite(3,HIGH);
                char ret = transmitUntilRecieved('L');
                Serial.println(ret);
                singDanceOrLight(ret);
                break;
            }
            else if (check == 2) {
                digitalWrite(2,HIGH);
                char ret = transmitUntilRecieved('K');
                singDanceOrLight(ret);
                break;
            }
            else if (check == 3) {
                digitalWrite(9,HIGH);
                char ret = transmitUntilRecieved('J');
                singDanceOrLight(ret);
                break;
            }
        }
        if (check == 4) break;
    }
}

char transmitUntilRecieved(char toTransmit) {
    servoLeft.detach();
    servoRight.detach();
    Xbee.begin(9600);
    while(1) {
        char charXbee = ' ';
        if(Xbee.available()) {
        charXbee = Xbee.read();
        }
        if(charXbee == 'X' || charXbee == 'Y' || charXbee == 'Z') {
            servoLeft.attach(13);
            servoRight.attach(12);
            return charXbee;
        }
        Xbee.print(toTransmit);
        //Serial.println(toTransmit);
        delay(100);
    }
}

void singDanceOrLight(char in) {
    if(in == 'X') {
        dance();
    }
    else if (in == 'Y') {
        servoLeft.detach();
        servoRight.detach();
        lightShow();
    }
    else if(in == 'Z') {
        sing();
    }
    servoLeft.detach();
    servoRight.detach();
}
```

```
181
182 void buzz(int targetPin, long frequency, long length) {
183     long delayValue = 1000000/frequency/2;
184     long numCycles = frequency * length/ 1000;
185     for (long i=0; i < numCycles; i++) {
186         digitalWrite(targetPin,HIGH);
187         delayMicroseconds(delayValue);
188         digitalWrite(targetPin,LOW);
189         delayMicroseconds(delayValue);
190     }
191 }
192
193 void sing() {
194     Serial.println(" 'Mario Theme'");
195     int size = sizeof(melody) / sizeof(int);
196     for (int thisNote = 0; thisNote < size; thisNote++) {
197         int noteDuration = 1000/tempo[thisNote];
198         buzz(melodyPin, melody[thisNote],noteDuration);
199         int pauseBetweenNotes = noteDuration * 1.30;
200         delay(pauseBetweenNotes);
201         buzz(melodyPin, 0,noteDuration);
202     }
203 }
204
205 void lightShow() {
206   pinMode(RLED, OUTPUT);
207   pinMode(GLED, OUTPUT);
208   pinMode(YLED, OUTPUT);
209   digitalWrite(RLED, LOW);
210   digitalWrite(GLED, LOW);
211   digitalWrite(YLED, LOW);
212   for(int k = 0; k < 5; k++) {
213     for(int j = 0; j < 5; j++) {
214       digitalWrite(RLED, HIGH);
215       delay(100);
216       digitalWrite(GLED, HIGH);
217       delay(100);
218       digitalWrite(RLED, LOW);
219       digitalWrite(YLED, HIGH);
220       delay(100);
221       digitalWrite(GLED, LOW);
222       delay(100);
223       digitalWrite(YLED, LOW);
224     }
225
226     for(int j = 0; j < 5; j++) {
227       digitalWrite(RLED, HIGH);
228       digitalWrite(GLED, HIGH);
229       digitalWrite(YLED, HIGH);
230       delay(200);
231       digitalWrite(RLED, LOW);
232       digitalWrite(GLED, LOW);
233       digitalWrite(YLED, LOW);
234       delay(200);
235     }
236   }
237 }
238
239 void dance() {
240     writeServo(1,-1);
241     delay(1000);
242     int count = 0;
243     while(count<3) {
244         count++;
245         writeServo(.5,.5);
246         delay(600);
247         writeServo(-.5,-.5);
248         delay(600);
249     }
250     count = 0;
251     while(count<3) {
252         count++;
253         writeServo(1,-1);
254         delay(600);
255         writeServo(-1,1);
256         delay(600);
257     }
258     delay(925);
259     writeServo(0,0);
260 }
```

Figure 1: Full Arduino Code

```
#define melodyPin 5
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE AS5 932
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978

//Mario main theme melody
int melody[] = {
  NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  NOTE_G7, 0, 0,  0,
  NOTE_G6, 0, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0,NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0,NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0
};
//Mario main them tempo
int tempo[] = {

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
};
```

Figure 2: Pitches.h Code

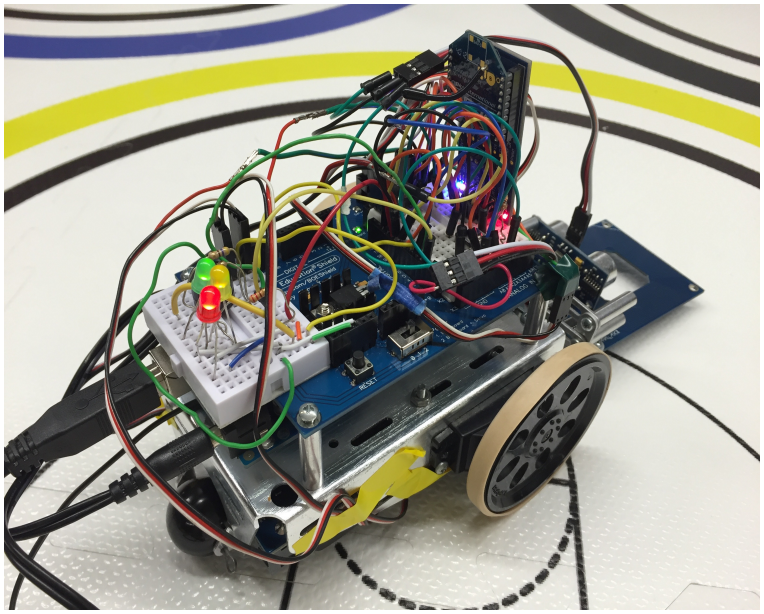

Figure 3: Photo of BOE-Bot (back)

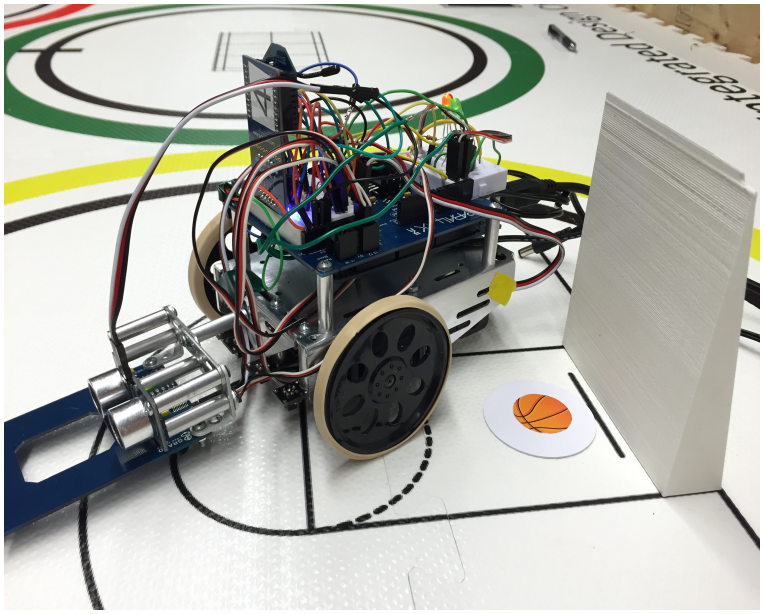

Figure 4: Photo of BOE-Bot (front)

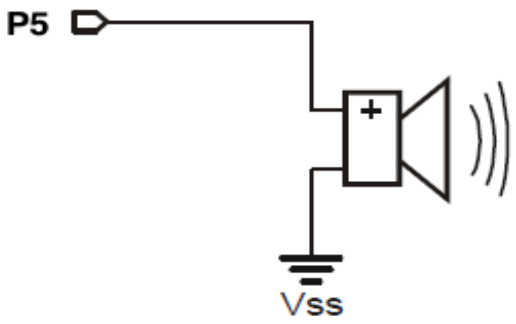

Figure 5: Piezo-speaker Schematic (learn.parallax.com)

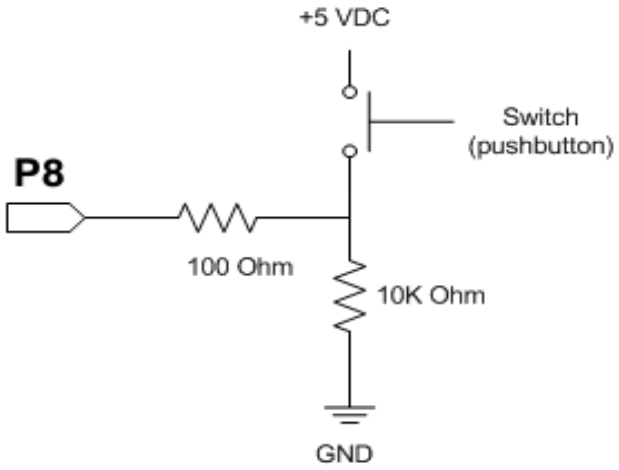

Figure 6: Push Button Schematic (http://blog.artificechicago.org)

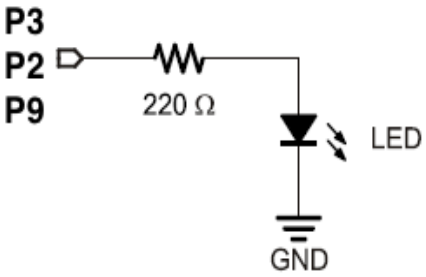

Figure 7: LED Schematic (learn.parallax.com)

References

[1] Kim, Michael, Juan Philippe, and David Tran. ECE 110L-01L: Conceptual Design Report - Final Draft. Durham, NC: Duke University, 2016.