ECE 110 Integrated Design Challenge: Final Report

Lab Section 01: Orange Squadron; Bot Box 38

Michael Casio -- mkc40
Stephanie Zelenetz -- sz119

We have adhered to the Duke Community Standard by completing this assignment.
December 9, 2017

ABSTRACT

This semester's Integrated Design Challenge (IDC) required five different robot squadrons to coordinate an assault on the Death Star. The task of the orange squadron was to design, build, and program a robot that could fulfill several tasks. The tasks of the orange squadron were to autonomously follow a black line, count the number of RFID tags present, communicate the amount of RFIDs detected to the "master bot", exit the first track in the appropriate order, and line up at the corresponding hash mark order down the ventilation shaft. To accomplish these tasks, the orange squadron used QTI sensors for both navigation and for RFID detection. Several technical challenges were present, mainly revolving around inconsistency in timing, overestimated timing and inappropriate wheel speeds, and inconsistent coordination with the other squadrons. Regarding software, the decision was made to create several functions and arrange the code in an organized and easy-to-read fashion. This allowed for easier debugging. In terms of performance, the orange squadron performed its task most of the time, with slight inconsistencies in sensing due to RFID placement. However, the team as a whole failed to complete tasks to their entirety; the red squadron consistently missed sensing, and was therefore unable to properly line up in the ventilation shaft. This, however, didn't hinder the rest of the groups, for the most part, since the red squadron was still able to communicate and allow other squadrons to proceed down the ventilation shaft, except for occasions in which it would block the path for other robots. This report provides details regarding the performance, successes, and challenges of the Orange squadron with regards to each task requested.

1. INTRODUCTION

1.1. Overview

During the Fall 2017 ECE 110L Integrated Design Challenge (IDC), each lab section was responsible for building robots that must work as a group to solve a multi-stage problem based on information they extract from the environment. This semester, the overall task was to defeat the Galactic Empire by destroying the second Death Star. As part of the Rebel Alliance, this lab's goal was to design and assemble Arduino BOE-Bots that would complete specific tasks in identifying and destroying the star, using task-specific simple sensors.

By dividing the lab group into five different squadrons, it became possible to divide the assault on the Death Star into smaller, more manageable tasks that can be completed with individual sensors or groups of simpler sensors. The design challenge encouraged students to develop skills to effectively present and communicate results, progress, and results throughout the project by weekly demos in which the entire class has to work together for a common goal, and weekly presentations to the class about the team's progress. This challenge reinforced the fundamental concepts of ECE 110. Students worked with their peers to apply basic circuitry techniques that enabled them to create a robot that accomplished the tasks at hand.

1.2. Team Objectives

Each individual group had a specific task that allowed it to contribute to the desecration of the Death Star.

The yellow squadron had to determine which of defensive turrets (located at each hash mark) are activated and report it to other squadrons. This was done by determining whether a turret's light is on. After evaluating which of the five turrets were on, the sum was calculated and reported to the other team. This the sum correlated to the position of the yellow squadron robot on the ventilation shaft.

The blue squadron had the responsibility of identifying the location of the Death Star. The Death Star was almost as large as the other moons of Endor. But unlike the other moons, it possessed a magnetic field and was gray. Both of these characteristics enabled the blue squadron to identify the Death Star. The blue squadron, thus, determined the location of the Death Star, reported it to the other squadrons. It, like the other squadrons, was required to proceed down the ventilation shaft in its proper order. Its final location (on the hash count) was the same number as the location of the Death Star.

The red squadron had the objective of identifying the number of TIE fighters and non-threatening ships that the Empire has available. This was done by determining which of the elements were white (TIE fighters), and which of the elements were black (non-threatening ships). The absolute value of the difference between the number of TIE fighters and non-threatening ships was computed and reported to the other squadrons. As with all the other squadrons, the red squadron proceeded down the ventilation shaft in its proper order, and stopped at the hash that corresponded to the the absolute value integer it computed.

The green squadron had the responsibility of determining the location of the entrance to the reactor shaft, and therefore the location of the optimal actact point. The security substations within the area of operation of this bot had five different colors, to determine the order in which the bots have to line up for the successful attack to the Death Star.

In this Integrated Design Challenge, the orange squadron had the objective of gaining access to the shield generator on one of Endor's moons in order to deactivate it. Its relay station broadcasted code to the imperial fleet from each of the five radio-frequency identification (RFID) tags. By adding the five unique signatures together, the orange squadron obtained the code to the shield generator. Once this was obtained and reported to the other robots, it was required to move down the ventilation shaft and stop at the proper location, which is the sum of the RFID signatures. This individual contribution to the success of the challenge can be broken down in the steps described in the next few sections.

1.3. Line-Following

The line-following challenge was to add and coordinate QTI sensors and motors in such a way that enabled the Arduino BOE-bot to follow a black line. In regards to the hardware, the BOE-bot included 4 QTI sensors to detect the difference between white and black on the IDC design board. A QTI sensor works by emitting an infrared signal through its built-in phototransistor. This reflects off a surface and return it to the black window of the QTI, which then strikes the infrared transistor's base of the QTI, causing it to conduct current. The more the incident on the transistor's base, the more the current it conducts. This in turn quantifies the amount of the infrared signal emitted is actually received. For the QTI to determine where it should move, the QTI sensors evaluates the reading it receives and decides whether it is higher or lower than a user-defined threshold. If it determines that the reading is lower than the threshold, it interprets as white, while a reading higher than the threshold is interpreted as black. Thus, when the values of the sensors read, from left to right, "white-black-black-white," it indicated to the robot that it was on track. Thus the robot proceeded forward. However, when the values of the middle sensors, from left to right, read "black-white" it meant that the robot was too much the right of the line, and would trigger the robot to self-correct and move a little to the left, and vice versa for the case where the middle sensors read "white-black." The robot would continue to self-correct until the QTI sensors were back to the "white-black-black-white" reading. Line following is important in that without it, it would be impossible to follow the track on the IDC design board and perform the rest of the challenge.

1.4. RFID Detection

The RFID detection challenge was to add and coordinate some sensor in such a way that the Arduino BOE-bot is able to detect the RFID signals. In this challenge, the BOE-bot utilized a QTI sensor (in addition to the four QTIs used for line-following) to detect RFID signals. Like line-following, the QTI sensor works by quantifying the brightness of the reflection of the

surface sensed by the built-in phototransistor. This QTI evaluated whether the RFID tag was black or white. A detection of "black" indicated that the relay station was transmitting a signal, while a detection of "white" indicated that the relay station was not active. The number of active relay stations was calculated. This enabled the robot to find the appropriate code to disable the shield generators. It also determined the robot's position when the final assault to the Death Star is performed.

1.5. Communication

The communication challenge was to find a way to communicate each group's appropriate integer value and to utilize that information to exit down the ventilation shaft in proper order, without overriding or repeating values. Because anything broadcast by the XBee module is broadcast to the entire room, robot to robot communications are not direct. In other words, all the robots would receive any signals sent. To solve this problem, the lab team developed a method to encode significance into unique characters. When certain characters were received, it signified a certain piece of data that a squadron is trying to convey. For example, a robot receiving four "y"s would indicate that the yellow squadron detected four active turrets. Characters irrelevant to a specific robot simply had no bearing on the robot's actions or calculations. Consequently, a string of characters could be simultaneously broadcast to the entire room, and only characters of significance to each robot would be stored and interpreted. This is important because without this step, the proper ordering of the robots cannot be determined as well as the values of each individual task.

1.6. Ventilation Shaft Exit

Like all the other squadrons, the orange squadron was tasked with the ability to exit through the ventilation shaft in proper order. After all robots communicated with each other and received the proper ordering, each individual robot made the jump towards the ventilation shaft, which was separate from the initial track the robot follows. This meant that each robot had to navigate through white space to get on a second track representing the ventilation shaft. This led to a slight change in the line-following algorithm, when it was implemented a second time. After all the robots received the proper order in which they must exit, a delay proportional to the robot's proper position begins, in order to prevent the robots from crashing into one another. The first robot exited immediately in order to increase efficiency.

To make the jump from one track to the other, the orange squadron robot was programmed to make a clockwise turn at a 45° and to go forward as the two middle QTIs detected "white-white." As it went forward, it started to detect the second track, and began to execute a line-following code similar to that of the first track. In this track, however, the robot was not programmed to stop at each hash mark and evaluate whether there is an RFID tag to its side. Instead, it was programmed to continue moving forward, while still counting hash marks, and stopping at the proper position (which is the integer value that it read from its sensing task).

This is important in that this is the final assault that destroys the Death Star. Only through properly exiting and lining up will the Empire be defeated.

2. EXPERIMENTAL PROCEDURE AND RESULTS

A stepwise approach to fulfilling the final task of the IDC was completed on a sub-task basis. In other words, the tasks that each bot needed to complete were broken up into weekly sub-tasks. This not only made the IDC manageable, but also easier. In the first week, individuals were tasked with ensuring that their robots were able to send and receive signals from one another. In the second week, they were tasked with line-following, in which their robots were required to follow four pre-printed lines, plus the actual line on the IDC design board that the squadron had to follow for the full integration demonstration. In the third week, students were tasked with following the line that their specific robot must follow for the full integration demonstration. They were also required to detect the objects for their individual sensing challenge (i.e. being able to sense the RFID tags.) In the fourth week, students had to ensure that their robots were able to integrate sensing and line-following. Specifically, the orange squadron had to travel down the line and stop at each hash mark to sense whether there was an RFID. In the fifth week, students were tasked with ensuring that all robots can simultaneously line-follow and sense the information present in their individual tasks. They also had to their individual data with the other robots. This task also included being able to display all information on an LCD display. In the sixth and last week of the IDC, groups not only were required to perform the tasks of week five, but also make the jump to the ventilation shaft and exit in proper order (i.e. perform the entire task of the integrated design challenge). Moving from one stage to the next, new circuits were added onto the existing circuit and new code was appended to the existing code in an accumulative fashion.

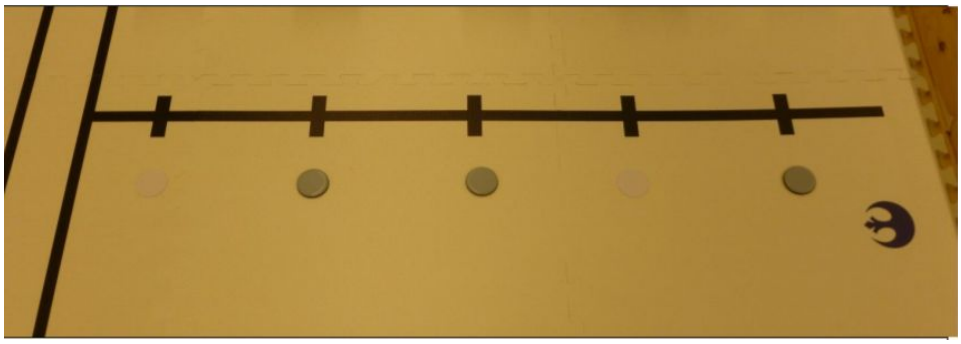

Figure 1: Orange Squadron Track needed to Navigate, and Sense RFID Tags

2.1. Week 1: Communication

The objective of the first week in this semester's IDC was to demonstrate the the robots could effectively and reliably send and receive signals from the other squadrons. To complete this task, an XBee module using the 9600-8-N-1 configuration was used because it

communicates with the Arduino microcontroller through serial ports that are easy to implement. Additionally, the wireless capability of the XBee module also satisfied the requirement to communicate with the other robots. The XBee modules operate as an asynchronous serial interface, meaning that data is transferred without the help of an external clock. This asynchronous serial protocol uses data bits, synchronization bits, and baud rate to avoid using an external clock. Synchronization bits indicate the beginning and end of a packet, and baud rate is how many bits per second are sent through the serial rate. In ECE 110, XBee modules with baud rates of 9600 were utilized.

The XBee modules function as a part of a hub and spoke network, in that it sends and receives transmissions to and from other nodes. For this week, to check communication, a sentry XBee robot was used to check for communication between the different bots in the lab. This sentry bot acts as a hub for which it sends and receives transmissions, while the lab's different squadrons act as spokes that also send and receive transmissions to both the sentry and the rest of the robots.

The XBee module that was plugged into the BOE-Bot is depicted in Figure 2.1.1. In this diagram, V_{ss} is connected to ground, while V_{dd} is connected to a +5V power. The TX and RX pins were wired based on the appropriate pins defined within the code. For this bot, the TX pin was connected to pin 11, while the RX pin was connected to pin 10, as seen in the code in Figure 2.1.4.

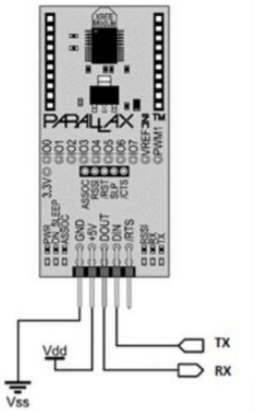

Figure 2.1.1: XBee Module Wiring

In addition to the actual XBee module, a push button as well as red and green LEDs were installed into the circuit. The circuit diagrams can be seen in Figures 2.1.2 and 2.1.3, respectively. These were installed to send signals (push button), and to show some sort of indication that the BOE-bot sent (red LED) and received (green LED) signals.

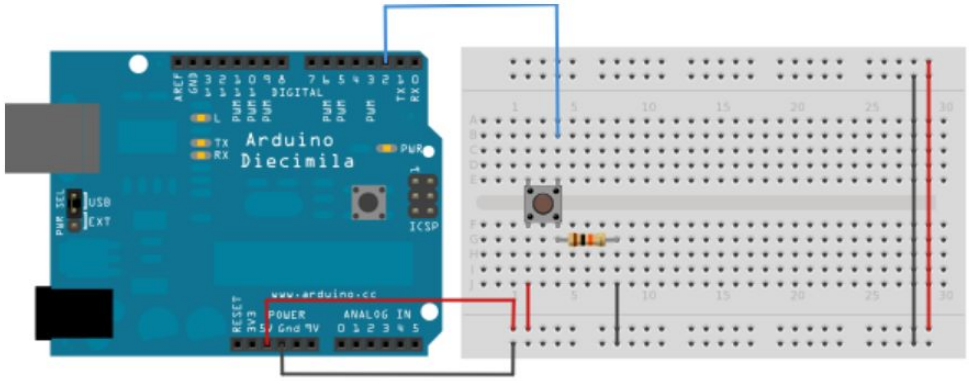

Figure 2.1.2: Wiring of the Push button

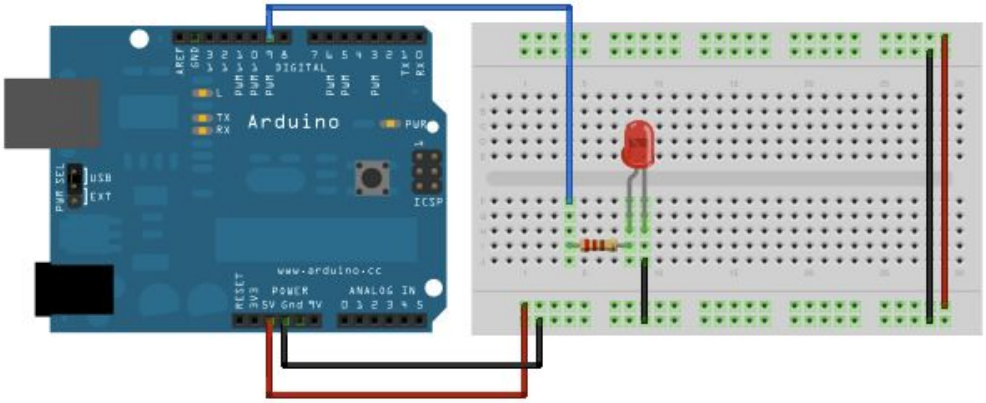

Figure 2.1.3: Wiring of the LEDs

As for the software aspect of the communication task, a scheme was established between the different squadrons. First, pins were defined for the push button and LEDs; the push button was set to pin 2, while the red LED to pin 8, and green LED to pin 9. Next, for the actual communication, the logic behind the code in Figure 2.1.4 was that if the push button was pressed, a character “g” would be sent out to the rest of the XBees modules around the room, including the sentry bot. When the character is sent out, the red LED would turn on for a second, then turn off, just to serve as an indication that a transmission signal was sent. Similarly, if the XBee module received some sort of signal, regardless of the character sent, the green LED would turn on for a second, then turn off.

This scheme proved to be very reliable and completed the task with ease. Although some LEDs were slightly dim and were hard to tell from a distance whether or not the XBee module sent/received transmissions, it still lit up in a manner that meant the different squadrons were able to communicate properly.

```
#include <SoftwareSerial.h>
#define Rx 10 // DOUT to pin 10
#define Tx 11 // DIN to pin 11
SoftwareSerial Xbee(Rx, Tx); // Xbee initialization

void setup() {
     Serial.begin(9600); // Set to no line ending
     Xbee.begin(9600); // types a char, then hits enter
     delay(500); // waits 0.5 seconds
     pinMode(8,OUTPUT); // initializes red outgoing LED
     pinMode(9, OUTPUT); // initializes green receiving LED
     pinMode(2, INPUT); // initializes button input
}
void loop() {
     if (digitalRead(2) == HIGH) { // if button is pressed
          char outgoing ='g'; // Read character, send to XBee
          Xbee.print(outgoing); // print to serial
          digitalWrite(8,HIGH); // turn on red outgoing LED
          delay(1000);
          digitalWrite(9,LOW); // turn off LED
          delay(500);
     }
     if (Xbee.available()) { // Is data available from XBee?
          char incoming = Xbee.read() ; // Read incoming character
          Serial.println(incoming); // send to Serial monitor
          digitalWrite(9, HIGH); // turn on green receiving LED
          delay(1000);
          digitalWrite(8,LOW); // turn off LED
     }
     delay (50) ; // reset
}
```

Figure 2.1.4: Communication Code

2.2. Week 2: Line-Following

The objective of the second week was to allow the robot to follow four pre-printed black lines, as well as the robot's track, and be able to automatically adjust its course if it travels astray. To achieve that task, four Parallax QTI sensors were installed in the front of the robot facing down, with two middle sensors being together, and two outer sensors about a quarter inch away from the middle sensors, as depicted in Figure 2.2.1. These sensors that are facing down have about a quarter inch distance between the sensors and the table surface.

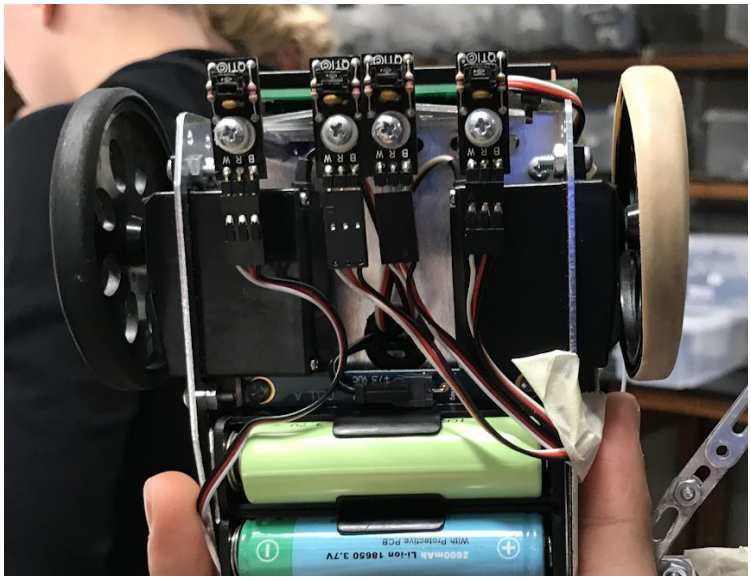

Figure 2.2.1: QTI Sensor Installation

The way that the QTI sensors were wired to the BOE-bot is depicted in Figures 2.2.2 and 2.2.3. The middle pin was connected to the digital pin on the BOE shield. For this week's task, this was connected to pins 7 (left QTI), 6 (left-middle QTI), 5 (right-middle QTI), and 4 (right QTI). These pins were used to control the charge and discharge of the capacitor inside the QTI sensors. The upper pin labeled with a W was connected to power, which was in the range of 3.3 V and 5 V. The lower pin labeled with a B was connected to ground.

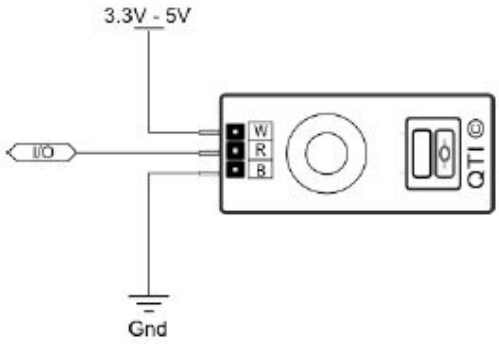

Figure 2.2.2: QTI Sensor Connection

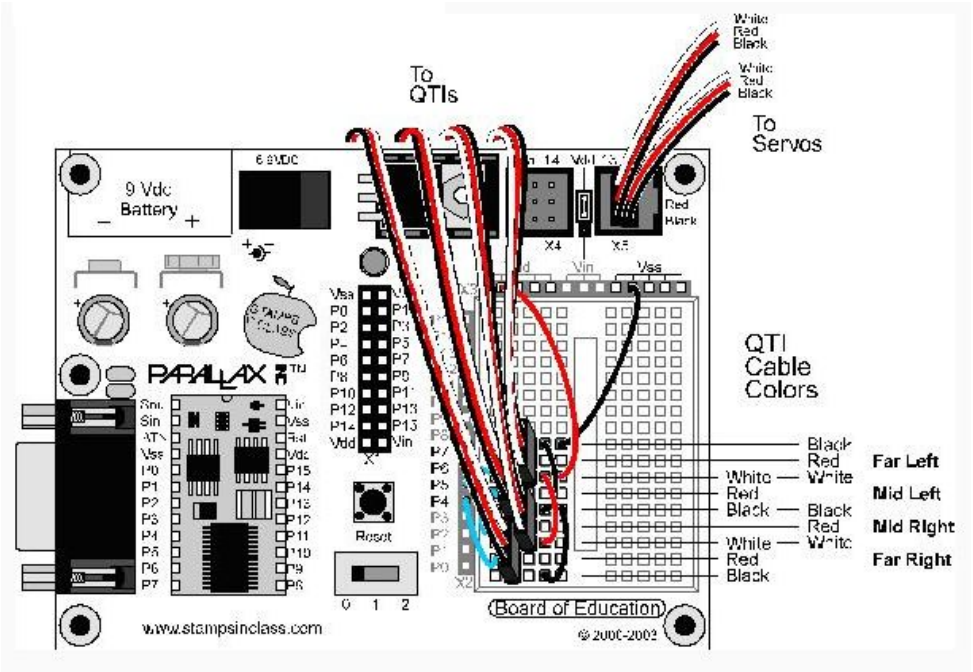

Figure 2.2.3: QTI Sensor Wiring

In the code, as depicted in Figure 2.2.4, all four QTI sensors were configured to charge for a certain amount of seconds until full. The QTI sensors then measured the time it took for the capacitor to fully discharge, which is the rate of charge transfer through the phototransistor implemented in the QTI. Depending on the reflectance of the surface that each QTI is facing, the capacitors would discharge to different extents, with those facing white surfaces discharging for a shorter amount of time, and returning a lower values. Thus, the rcTime function returned shorter values for white surfaces and higher values for black surfaces. To ensure that line-following works properly, a threshold of 500 was determined by experimentation. The QTIs sensed white if the rcTime was lower than 500, and sensed black if the rcTime was greater than 500. So, cases were set to which what the robot would do if its QTIs sensed black or white. If the two middle cases sensed black, the robot continued to move forward since that means that the robot is in the right track. If the left-middle QTI sensed black, while the right-middle QTI sensed white, the robot was slightly to the right, causing the robot to self-correct by moving a little to the left to get back on track. This is similar for the case in which the right-middle QTI sensed black, while the left-middle QTI sensed white; it meant that the robot was slightly to the left, and thus the robot would self-correct its course by moving a little to the right. If the outer QTIs detected black, the robot has reached a hashmark. In this case, the robot paused for a short amount of time, before continuing to move forward for half a second until it entered the top of the loop once again.

It was observed that the robot moved rather twitchy in early tests. To allow a smooth line following, different combinations of wheel rotation speed and spacings between the four QTI

sensors were tested. In addition to the twitchy movement of the robot in earlier tests, the robot struggled to make a 90° turn when it moved fast. For these reasons, the best combination was determined to be a slightly moderate speed for when the robot was making a turn, and a slightly faster speed when the robot was moving straight. The sensor location was optimized when the middle sensors were as close together as possible and the outer sensors about a quarter inch away from the middle sensors.

```
void loop() {
  long qti1 = rcTime(7); // finds reading of left QTI using rcTime function
  long qti2 = rcTime(6); // finds reading of left-mid QTI
  long qti3 = rcTime(5); // finds reading of right-mid QTI
  long qti4 = rcTime(4); // finds reading of right QTI


  if (qti2 > 500 && qti3 > 500) { // if two mid QTIs see black
    servoLeft.writeMicroseconds(1470);       // go forward
    servoRight.writeMicroseconds(1530);
    delay(50);
  }
  else {
    if (qti2 > 500 && qti3 < 500) { // if left-mid QTI sees black, but not right-mid QTI
      servoLeft.writeMicroseconds(1440); // self-correct by going to the left a little
      servoRight.writeMicroseconds(1440);
    }
    if (qti2 < 500 && qti3 > 500) { // if right-mid QTI sees black, but not left-mid QTI
      servoLeft.writeMicroseconds(1560); // self-correct by going to the right a little
      servoRight.writeMicroseconds(1560);
    }
  }
  if (qti1 > 500 && qti4 > 500) { // if the outer QTIs see black, means at a hash mark
    servoLeft.writeMicroseconds(1500); // pause for a second
    servoRight.writeMicroseconds(1500);
    delay(1000);
    servoLeft.writeMicroseconds(1450); // continue moving forward for half a second
    servoRight.writeMicroseconds(1550); // enough movement to not read hash mark twice
    delay(500);
  }
}


 long rcTime(int pin)
 {
   pinMode(pin, OUTPUT);                  // charge capacitor
   digitalWrite(pin, HIGH);               // ..by setting pin ouput-high
   delay(1);                              // ..for 5 ms
   pinMode(pin, INPUT);                   // Set pin to input
   digitalWrite(pin, LOW);                // ..with no pullup
   long time  = micros();                 // Mark the time
   while (digitalRead(pin));              // Wait for voltage < threshold
   time = micros() - time;                // Calculate decay time
   return time;                           // Return decay time
 }
```

Figure 2.2.4: Line-following code

2.3. Week 3: Information Gathering

For the third week's challenge, the robot had to continue to line-follow as it did in the previous week, sense the RFID tag, and provide some sort of indication that it can sense such tag. To achieve the task, the previous group that was assigned the orange squadron utilized an

RFID sensor to detect RFID tags. Although they were able to get it to detect properly and to have an indication that they did detect a tag properly, a QTI sensor was ultimately used to complete this task. This is due to a few reasons. First, when the robot was passed on group, the RFID sensor stopped working and would not detect tags. It was initially thought that it was a problem with the sensor. So a new sensor was requested. However, after a few hours of unsuccessfully experimenting with the new sensor, it was decided that it would be best to switch sensors. A QTI was chosen because it was already known to be reliable, and previous QTI code had already been written for the line-following.

Furthermore, one of wires connecting the RFID sensor the previous group installed go unplugged, and it was difficult to determine where that wire belonged or whether or not it was plugged in properly. For these reasons, a fifth QTI was added to the robot and an arm was added for sensing, as depicted in Figure 2.3.1. Additionally, since the RFID tags had both a white side and a black side, the QTI's ability to differentiate a black surface and a white surface benefitted us in terms of being able to detect whether or not there is a QTI present.

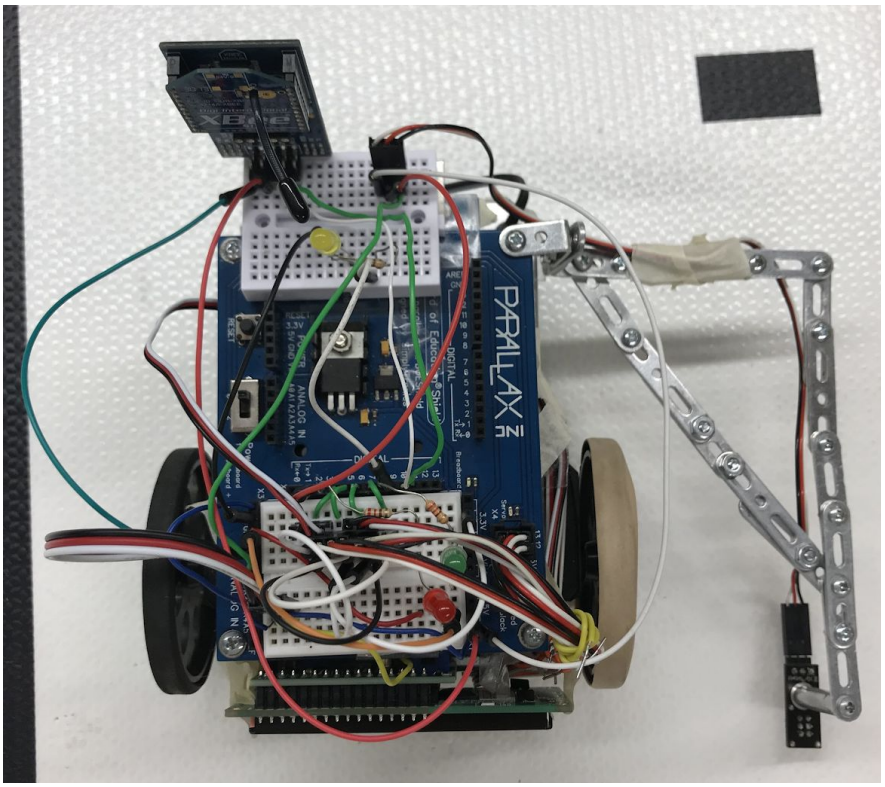

Figure 2.3.1: Arm and Fifth QTI added to the Orange Squadron Robot

In regards to code, a new QTI was initialized and rcTime values were measured at the top of the loop, as depicted in Figure 2.3.2.

```
void loop() {

  long qti1 = rcTime(7);
  long qti2 = rcTime(6);
  long qti3 = rcTime(5);
  long qti4 = rcTime(4);
  long qtiRFID = rcTime(2);
```

Figure 2.3.2: RFID using QTI Code Added for Week 3

2.4. Week 4: Integrated Sensing, Processing, & Navigation

The objective of the fourth week was to be able to integrate sensing, line-following, and displaying of the information gathered. To achieve this task, both the line-following and the sensing code was integrated into the same code. In regards to hardware, little was changed from that of the previous week. Slight modifications were made to the arm (e.g. making the arm a bit longer/more stable). However, most changes were changes to the code.

First and foremost, a subtask for the fourth week was to be able to get the robot to line-follow, stop at each hash mark, and stop at the end of the line, just before the jump to the ventilation shaft. To achieve this subtask a counter was set in that it incremented each time the robot reached a hash mark. When the robot reached the sixth hash mark, it would stop indefinitely and display the amount of RFIDs it detected, as depicted in Figure 2.4.1.

Second, the code was modified in that for every hash mark that isn't the sixth hash mark, the QTI that hovers over the RFID started to read and check whether an RFID is present or not. Additionally, it was important that the robot was able to keep track of the number of RFIDs that it detected. To achieve this, a counter for RFIDs was set, and for each time that the robot stops at a hash mark and detects an RFID, it increments the RFID counter. Again, this can be seen in Figure 2.4.1.

Lastly, to display that the robot sensed properly, a yellow LED was installed similar to the one depicted in Figure 2.1.3. The yellow LED would light up for a second and then turn off after each time an RFID was detected at the individual hash marks.

```
#include <SoftwareSerial.h>
#define Rx 10 // DOUT to pin 10
#define Tx 11 // DIN to pin 11
#define LCDPin 53
SoftwareSerial Xbee(Rx, Tx); // Xbee initialization
#include <Servo.h>                          // Include servo library
Servo servoLeft;                            // Declare left and right servos
Servo servoRight;
SoftwareSerial mySerial = SoftwareSerial(255, LCDPin);
int hashCount = 0;
int RFIDcount = 0;

#include <SoftwareSerial.h>

void setup() {
     Serial.begin(9600); // Set to no line ending
     delay(500); // waits 0.5 seconds
     pinMode(3,OUTPUT); // initializes red outgoing LED
     pinMode(8, OUTPUT); // initializes green receiving LED
     pinMode(9,OUTPUT); // initializes yellow LED for sensing
     servoLeft.attach(13); // Attach left signal to pin 13
     servoRight.attach(12);  // attach pin 12 to right servo
     digitalWrite(9, LOW);
}

void loop() {

     long qti1 = rcTime(7); // finds reading of left QTI using rcTime function
     long qti2 = rcTime(6); // finds reading of left-mid QTI
     long qti3 = rcTime(5); // finds reading of right-mid QTI
     long qti4 = rcTime(4); // finds reading of right QTI
     long qtiRFID = rcTime(2); // finds reading of RFID QTI


 if(qti2>500 && qti3>500){ // if two mid QTIs see black
     servoLeft.writeMicroseconds(1470); // go forward
     servoRight.writeMicroseconds(1530);
     delay(50);
 }
 else {
     if(qti2>500 && qti3<500){ // self-correct by going to the left a little
         servoLeft.writeMicroseconds(1440); // self-correct by going to the left a little
         servoRight.writeMicroseconds(1440);
     }

     if(qti2<500 && qti3>500){ // if right-mid QTI sees black, but not left-mid QTI
        servoLeft.writeMicroseconds(1560); // self-correct by going to the right a little
        servoRight.writeMicroseconds(1560);
     }
 }

 if (qti1>500 && qti4>500) { // if the outer QTIs see black, means at a hash mark
     servoLeft.writeMicroseconds(1500);// pause for a second
     servoRight.writeMicroseconds(1500);
     hashCount++; // increment hash count

     if (hashCount ==6){ // if the robot is at the end of track
         servoLeft.writeMicroseconds(1500); // stop there
         servoRight.writeMicroseconds(1500);
         delay(1000);
     }
     else{

delay(1000);
qtiRFID = rcTime(2); // sets up the rfid on the side; logic is that if counter isn't 6, then scan the rfid on the side
qtiRFID = rcTime(2); // reads again for good measure
if (qtiRFID > 500){
    RFIDcount++;
    digitalWrite(9,HIGH); // if the rfid is black, then light up the yellow led, if not then don't turn it on
    delay(1000);
    digitalWrite(9,LOW); // turn the yellow led back off
}

servoLeft.writeMicroseconds(1450);         // keep going after reading the rfid
servoRight.writeMicroseconds(1550);
delay(500); }
}
}
```

Figure 2.4.1: Code for Integrated Line-Following and Sensing

2.5. Week 5: Team Sensing, Processing, & Navigation

For the fifth week, the main challenge was to perform the challenge from the previous weeks, while also communicating with the other robots the information gathered and displaying such information in an LCD serial display. To do this, an LCD serial display was installed to the robot, as shown in Figures 2.5.1, 2.5.2, and 2.5.3.

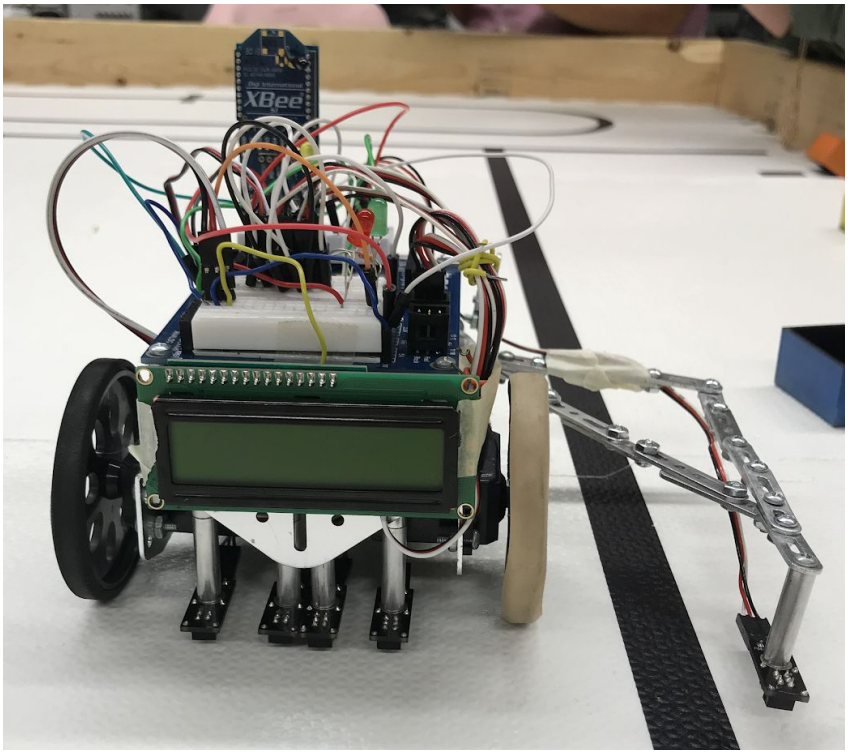

Figure 2.5.1: Front View of Robot with LCD Display

For the LCD Display, the middle pin V_{dd} is connected to power, which in this case is 5V, while the upper pin is connected to the pin dedicated for the LCD, which in this case is pin 53. The lower pin V_{ss} is connected to ground.

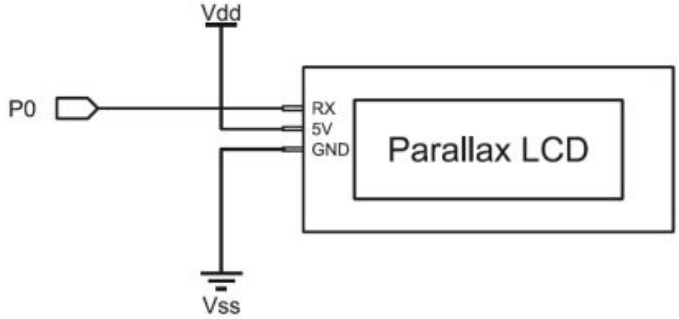

Figure 2.5.2: Wiring of LCD Serial Display

```
#include <SoftwareSerial.h>
#define Rx 10 // DOUT to pin 10
#define Tx 11 // DIN to pin 11
#define LCDPin 53
SoftwareSerial Xbee(Rx, Tx); // Xbee initialization
#include <Servo.h>                          // Include servo library
Servo servoLeft;                            // Declare left and right servos
Servo servoRight;
SoftwareSerial mySerial = SoftwareSerial(255, LCDPin); // initializes LCD Display
int hashCount = 0; // keeps track of hash marks
int RFIDcount = 0; // keeps track of number of RFID
int turretCount = 0; // keeps track of yellow squad info
int deathCount = 0; // keeps track of blue squad info
int TIEcount = 0; // keeep track of red squad info
int blockCount = 0; // keep track of green squad info

#include <SoftwareSerial.h>

void setup() {
  Serial.begin(9600); // Set to no line ending
  Xbee.begin(9600); // types a char, then hits enter
  delay(500); // waits 0.5 seconds
  pinMode(3, OUTPUT); // initializes red outgoing LED
  pinMode(8, OUTPUT); // initializes green receiving LED
  pinMode(53, OUTPUT); // initializes LCD output
  pinMode(9, OUTPUT); // initializes yellow LED for sensing
  servoLeft.attach(13); // Attach left signal to pin 13
  servoRight.attach(12);  // attach pin 12 to right servo
  digitalWrite(9, LOW); // sets yellow LED to low
  digitalWrite(LCDPin, HIGH); // turns on LCD display
  mySerial.begin(9600); // initializes LCD display
  delay(100);
  clearLCD(); // clears LCD
  delay(5);
  digitalWrite(LCDPin, LOW); // sets LCD display to low

}

while (v < 50000) { // enter while loop
  xbeeReceive(); // receive
  mySerial.print("Y:"); // display values as they come in on the LCD display
  mySerial.print(turretCount);
  mySerial.print(", B:");
  mySerial.print(deathCount);
  mySerial.print(", O:");
  mySerial.print(RFIDcount);
  newLine(); // starts a new line the LCD
  mySerial.print("R:");
  mySerial.print(TIEcount);
  mySerial.print(", G:");
  mySerial.print(blockCount);
  newLine();
  v++; // increment counter
}
attachServos(); // after receiving, reattach servos
delay(5000); // delay for five seconds
}
```

Figure 2.5.3: Code for LCD Display Setup and Displaying

Furthermore, as depicted in Figure 2.5.4, an algorithm was set in that a counter for each of the other robots' tasks was initialized (e.g. setting a counter for the number of active turrets, the location of the death star, etc). A "master robot" was established, in which it would reach the

end of the first track first and start receiving signals from the other bots. In our lab, the yellow squadron was the master robot. The other robots would slowly make its way through its individual track and sense its objects as it did in the previous week. Once it reached the end of its track, it would enter a loop in which it would send out the appropriate amount of lowercase characters to the master robot to communicate the number of objects it detected. For example, if our orange squadron detected four RFIDs, once the robot reached the end of the track, it would enter a loop in which it would send out four “o”s. Only the master bot would have a code programmed in to interpret the lowercase “o” character, so that the other robots that may be at the end of the track don’t interpret the character and double count later on. After the appropriate number of “o”s was sent, the orange robot would send out a single lowercase “d” to indicate to the master robot that it was done transmitting “o”s. The master robot continued to receive lowercase letters until it received four “d”s. This indicated that the other four robots were done with its transmission. Then, the master bot would break out of its receiving loop and relay everyone’s information by sending out the appropriate number of uppercase letters corresponding to the color of each robot (e.g. if it received four lowercase “o”s, it would send out four uppercase “O”s). Again, this system of character was important so double counting could be avoided. The “d”s are extremely crucial in that if the master robot didn’t receive it, it wouldn’t break out of its receiving loop and no robot would receive any information. After receiving all the information, each robot displayed the information on its LCD display.

```
void xbeeReceive() {
  if (Xbee.available()) { // Is data available from XBee?
    char incoming = Xbee.read() ; // Read incoming character
    if (incoming == 'Y') { // if receive Y, add one to turret count
      turretCount++;
    }
    if (incoming == 'B') { // if receive B, add one to death star count
      deathCount++;
    }
    if (incoming == 'R') { // if receive R, add one to tie fighter count
      TIEcount++;
    }
    if (incoming == 'G') { // if receive G, add one to block count
      blockCount++;
    }
    digitalWrite(8, HIGH); // turn on green receiving LED
    delay(100);
    digitalWrite(8, LOW); // turn off LED
  }
  delay (5) ; // reset
}
```

Figure 2.5.4: Code for Receiving Algorithm

As for strategy and reasons why the lab chose the route of having a master robot take in all the information, we recognized that all of the robots had different speeds in proceeding down its line and sensing its element. For this reason, a robot could start sending early when no one

was ready to receive information (because it is still in the middle of line-following). This means that signals could be lost and not counted. Furthermore, it was observed that XBee communication interferes with the servo motors. Even if the robot was executing the line-following part of the code, if it receives a signal, the robot would jump to the XBee code and interfere with the movement of the servos. Having a master robot allowed us to bypass these issues, as well as issues of double counting signals.

To implement receiving the appropriate characters, an if-tree was implemented within the XBee receiving code, in that if the robot receives a certain character, a counter for the specific character is incremented, as depicted in Figure 2.5.4. This allowed the robot to keep track of all the other robots' information.

One aspect to which we had trouble with was that the XBee module would interfere with the servos even when the servos aren't moving anymore. For example, when the robot would reach the end of the line and start receiving, the servos would cause the robot to move in circles, even though it wasn't coded to do that. To solve this, we detached the servos before the robot started receiving signals from the master robot and reattached them after it was done receiving all information.

2.6. Week 6: Full System Integration

For the sixth and final week of the IDC, teams were required to perform the same tasks as the previous week, while also being able to jump over the white space to the second U-shaped ventilation shaft track. This final task required proper timing, for each bot had to exit in the appropriate order and stop at the appropriate hash mark (e.g. if the orange squadron sensed 1 RFID, it would be the first to exit; if it sensed 3 RFIDs, then it would be the third to exit). This required that all robots completed all individual tasks successfully and were able to perform the last jump to the second track.

To do this task and to prevent all of the robots from colliding each other, the master robot sent out an uppercase "D" after sending out all of the robots' informations. After receiving this character, each robot waited for 30(n-1) seconds, with n representing number that they obtained through their initial sensing task. This ensured that each robot had enough time jump the white space and make its way down the line. The n-1 factor was implemented so that the robot that was supposed to exit first did not have to wait for 30 seconds before making the jump.

To get the robot to jump, a piece of code that makes the robot rotate 45° clockwise and move forward for a bit was implemented. Next, the robot entered a new line-following loop that had the code that if the middle QTIs sense "white-white" then it would go straight. This ensured that the robot kept on going straight until it hit the second track, where it self-corrected through the previous line-following code and line-follow as programmed. Another modification, as seen in Appendix A, was that the robot no longer stopped at a hash mark to evaluate an RFID to its side since there were no more RFIDs to evaluate. Instead, the robot moved past the hashmark,

with a new hash count, and stopped at the appropriate mark, as seen in Figure 2.6.1. To get the robot to properly make the jump, the speeds of the servos were experimentally tested.

A big challenge was making sure that the robots waited the appropriate amount of time before it made the jump to the next track. One of the problems faced was that if the robot was in the fourth position and was supposed to wait a minute and a half before making the jump, it would move much earlier. This was found to be the result of setting the delay value as an integer because arduino isn't able to store an integer delay of more than 30000 ms. To resolve this the value of the delay was typecasted to become a float, which the delay function can handle in large quantities.

The biggest challenge for the overall system run was mainly getting the robot to make the jump smoothly and reliably. Initially, it was hard coded into the robot to make the jump. However, it was realized that this caused the robot to make wide turns, which resulted in collisions with robots that may have been present in the third hash mark. For this reason a 45 degree turn was implemented instead of heading straight to the second track. But, this required experimentation in terms of wheel speeds and angles to make sure that the robot did not go past the track.

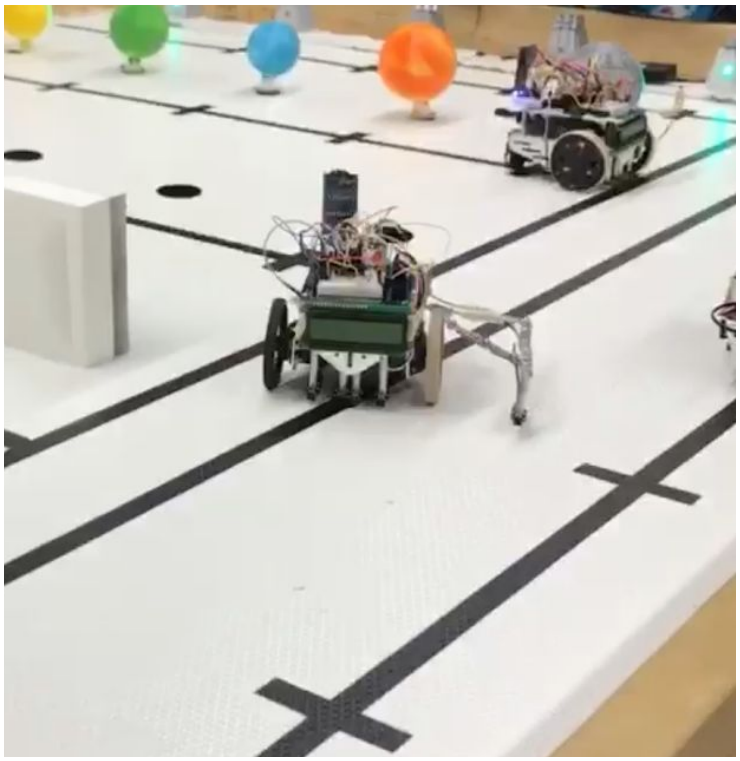

Figure 2.6.1: Orange Bot Making Full System Task

3. ANALYSIS AND DISCUSSION

3.1 Statement of Cost

The total cost of the bot was approximately 310.69 dollars, which was fairly low. The cost of the QTI was significantly lower than some of the other sensors that could be used to detect the RFID tags, such as the RFID Reader Module. However, it was still more expensive than some of the other sensors that could be used to complete the task. Thus, the cost could be minimized by choosing a different sensor to detect the RFID discs, such as the IR sensor and LED pair.

3.2 Analysis of Final Design

The final design of the bot was highly effective, for it enabled the bot to complete the majority of tasks outlined in the Experimental Procedures and Results section.

The systems design allowed for highly effective line-following. The bot was designed to include four QTI sensors that would allow it to sense its relative location on the board, and proceed accordingly. Specifically, the bot contained four QTI sensors situated on the bottom front of the bot. There was minimal spacing between the QTIs, which enabled for a higher degree of precision. The choice to use four QTIs (rather than three or two) was a highly effective one, for it allowed for a greater degree of accuracy when line following. With a greater number of QTIs, more information about the bot's relative location on the board (i.e. if it was on a hash mark) was accessible, so line following could be executed with a greater accuracy and efficiency. The two middle QTIs were designed to be situated on the line (unless the bot veered off course), while the outer two straddled the line. If the bot started to move off to one side or the other, one of the middle QTIs would sense a white, rather than black, surface. This information would be relayed to the system, and an appropriate correction would be made. If only three sensors were used, with one initially straddling the center of the line, and the two others on the periphery, there would be less accuracy. For instance, if the bot started to move at an angle, it likely would take longer for this information to be relayed for the system and the appropriate adjustment to me made. Thus, the bot would move more off course before the correction was made, and the correction would have to be more extreme. While the line following was fairly effective, the speed of the bot could have been optimized, as could the ease with which it completed turns.

A QTI sensor was also used to detect the RFID tags. This proved to be effective in that when the QTI sensor was in line with the RFID tag, it detected the tag with complete accuracy. The only issue was the alignment of the QTI. Because the RFID tags were so far away from the hash marks the bot stopped at during detection, the QTI sensor could not be mounted onto the side of the bot. As a result, an arm was constructed that would enable the QTI to extend from the bot to hover over, and eventually sense, the RFID tag. The issue with this was that the QTI only possesses a small detection area. If the alignment of the QTI and the RFID tag was not perfect, the QTI would not appropriately detect the tag. It was difficult to achieve this optimal alignment

because the bot stopped at slightly different spots on the line each time. This is something that will be discussed in the Proposed Changes Section.

The choices to use the XBee for communication was highly effective. The bot correctly sent and received all information during each of the five trials. The use of the serial LCD to display the information communicated was also successful.

3.3 Proposed Changes

Many improvements could have been made to optimize the bot's performance and efficiency given additional time and resources. First, the bot's line following could have been optimized. As a whole, the bot's line following was fairly effective. It had no issues following straight lines, but the speed at which is followed the line could have been increased. It was not increased during the final stages of the project because increasing the speed could have potentially decreased the reliability of the bot. During the final demonstration, the bot's line-following was strong, but not perfect. It did not complete one of the turns during the demo, and the turns it did complete successfully were not smooth. This could be optimized through trial and error: by changing the wheel speed of the servos until the line following is smooth and reliable. The sensing of the bot could also be improved. In order to sense the RFID tags an arm was built off of the bot. At the end of this arm was a QTI sensor that hovered over the RFID tag and sensed the appropriate signal. The issue with this was that the alignment was inconsistent. Because only a small area of the QTI did the sensing, and the location at which the bot stopped on the hash was inconsistent, it was hard to ensure that the QTI would always be hovering over the RFID tag to sense it correctly. Perhaps measurements could be taken and the arm could be adjusted so that the probability of the sensing part of the QTI being aligned with the RFIDs at each hash would be increased.

4. CONCLUSION

This project was highly successful. While the bot's performance was not flawless, it executed line-following, RFID-detection, communication, and the ventilation shaft exit extremely well. The bot's only issue during the demo was with the line following during the ventilation shaft exit. The bot followed the line and appropriately stopped at hash marks during the first part of the challenge. The issue with the line following arose during the U-shaped turn during the ventilation shaft exit. During one of the five trials the bot ran off course during the final part of the turn, and was unable to finish the trial. This loop had not posed an issue during earlier weeks. It is likely that the issue arose when the wheel speeds of the bot were altered in order for it to successfully "jump" from the first part of the course to the second. As a result, the robot would perform sharper turns when it was self-correcting, thus resulting in it veering off-course.

The bot's RFID sensing worked perfectly during all of the five trials. While the issue of alignment proposed an issue, trial runs before the actual demo allowed for the ideal placement of the RFID tags to be determined. This was a great success.

The communication was also entirely successful. The bot sent the correct sensing information to the "master bot" and received the appropriate information as well. This was certainly one of the most team-oriented parts of the challenge. The way the part of this challenge was coded required that all bots appropriately send out their sensing information to the "master bot" in order to proceed onwards. Thus, if one bot did not send out the sensing information, none of the bots would be able to complete the ventilation shaft exit. Luckily all bots communicated with the master bot. This was one of the greatest successes of the day.

The greatest challenge of the day arose when one of the other bots did not sense correctly. This caused it to complete the ventilation shaft exit at the incorrect time and during the first trial, caused a mass pile-up of bots. A group-wide decision was made to remove the bot from the course if it exited incorrectly. While this situation was not ideal, it exemplifies one of the strengths of the lab section as a whole: the ability to communicate and work together effectively.

The bot successfully completed the ventilation shaft exit, which was a concern during testing. This proved to be another great success for the lab team.

Overall, the completion of the Integrated Design Challenge was highly successful. There were many different aspects of the course that came with their own sets of challenges. Ironically, the greatest strengths and weaknesses of the day entailed the ability to work together as an entire lab section to complete group-oriented tasks. The bot's performance was highly accurate, though not flawless. It would have been hard to achieve perfection, for their are so many factors that come into play, and so many variables that are almost out of one's control.

APPENDIX A: FULL CODE

```
#include <SoftwareSerial.h>
#define Rx 10 // DOUT to pin 10
#define Tx 11 // DIN to pin 11
#define LCDPin 53
SoftwareSerial Xbee(Rx, Tx); // Xbee initialization
#include <Servo.h>                          // Include servo library
Servo servoLeft;                            // Declare left and right servos
Servo servoRight;
SoftwareSerial mySerial = SoftwareSerial(255, LCDPin); // initializes LCD display
int hashCount = 0; // keeps track of hash marks
int RFIDcount = 0; // keeps track of number of RFID
int turretCount = 0; // keeps track of yellow squad info
int deathCount = 0; // keeps track of blue squad info
int TIEcount = 0; // keeep track of red squad info
int blockCount = 0; // keep track of green squad info
int newHash = 0; // keeps track of second track's hash count
int bigD = 0; // keeps track of whether D hs been received
float T = 0; // initializes delay for jump
#include <SoftwareSerial.h>

void setup() {
  Serial.begin(9600); // Set to no line ending
  Xbee.begin(9600); // types a char, then hits enter
  delay(500); // waits 0.5 seconds
  pinMode(3, OUTPUT); // initializes red outgoing LED
  pinMode(8, OUTPUT); // initializes green receiving LED
  pinMode(53, OUTPUT); // initializes LCD output
  pinMode(9, OUTPUT); // initializes yellow LED for sensing
  servoLeft.attach(13); // Attach left signal to pin 13
  servoRight.attach(12);  // attach pin 12 to right servo
  digitalWrite(9, LOW); // sets yellow LED to low
  digitalWrite(LCDPin, HIGH); // turns on LCD display
  mySerial.begin(9600); // initializes LCD display
  delay(100);
  clearLCD(); // clears LCD
  delay(5);
  digitalWrite(LCDPin, LOW); // sets LCD display to low

}

void loop() {

  long qti1 = rcTime(7); // finds reading of left QTI using rcTime function
  long qti2 = rcTime(6); // finds reading of left-mid QTI
  long qti3 = rcTime(5); // finds reading of right-mid QTI
  long qti4 = rcTime(4); // finds reading of right QTI
  long qtiRFID = rcTime(2); // finds reading of RFID QTI
```

```
if (qti2 > 500 && qti3 > 500) { // if two mid QTIs see black
  servoLeft.writeMicroseconds(1470); // go forward
  servoRight.writeMicroseconds(1530);
  delay(50);
}
else {
  if (qti2 > 500 && qti3 < 500) { // if left-mid QTI sees black, but not right-mid QTI
    servoLeft.writeMicroseconds(1440); // self-correct by going to the left a little
    servoRight.writeMicroseconds(1440);
  }

  if (qti2 < 500 && qti3 > 500) { // if right-mid QTI sees black, but not left-mid QTI
    servoLeft.writeMicroseconds(1560); // self-correct by going to the right a little
    servoRight.writeMicroseconds(1560);
  }
}

if (qti1 > 500 && qti4 > 500) { // if the outer QTIs see black, means at a hash mark
  servoLeft.writeMicroseconds(1500); // pause for a second
  servoRight.writeMicroseconds(1500);
  hashCount++;

  if (hashCount == 6) { // if at end of first track, which is hashmark 6
    servoLeft.writeMicroseconds(1500); // stops
    servoRight.writeMicroseconds(1500);
    delay(1000);
    for (int s = 0; s < RFIDcount; s++) { // send appropriate number of o's
      xbeeSend('o');
    }
    xbeeSend('d'); // send one d
    delay(50);
    detachServos(); // detach servos to prevent bot from going crazy
    int v = 0;
    while (v < 50000) { // enter while loop
      xbeeReceive(); // receive
      mySerial.print("Y:"); // display values as they come in on the LCD display
      mySerial.print(turretCount);
      mySerial.print(", B:");
      mySerial.print(deathCount);
      mySerial.print(", O:");
      mySerial.print(RFIDcount);
      newLine(); // starts a new line the LCD
      mySerial.print("R:");
      mySerial.print(TIEcount);
      mySerial.print(", G:");
      mySerial.print(blockCount);
      newLine();
      v++; // increment counter
      delay(100);
      if (bigD > 0) { // if D received, break out of this loop
```

```
    break;
  }
}
T = 30000 * float(RFIDcount - 1); // sets delay for jump
delay(T);

attachServos(); // reattach servos
servoLeft.writeMicroseconds(1700); // makes 45 degree clockwise turn
servoRight.writeMicroseconds(1510);
delay(1200);
servoLeft.writeMicroseconds(1440); // go straight for 1875 ms
servoRight.writeMicroseconds(1560);
delay(1875);

while (1) { // second line-following
  long qti1 = rcTime(7); // finds reading of left QTI using rcTime function
  long qti2 = rcTime(6); // finds reading of left-mid QTI
  long qti3 = rcTime(5); // finds reading of right-mid QTI
  long qti4 = rcTime(4); // finds reading of right QTI

  if (qti2 > 500 && qti3 > 500 && qti1 < 500 && qti4 < 500) { // if mid QTIs black
    servoLeft.writeMicroseconds(1420); // go forward
    servoRight.writeMicroseconds(1580);
    delay(50);
  }
  if (qti1 > 500 && qti4 > 500 && qti2 > 500 && qti3 > 500) { // if sense black hash
    digitalWrite(9, HIGH); // light up LED
    delay(100);
    digitalWrite(9, LOW);
    newHash++; // increment new hash count
    if (newHash == (6 - RFIDcount)) { // if you find the place to stop
      detachServos(); // stop moving
      mySerial.print("Y:"); // display all squad info
      mySerial.print(turretCount);
      mySerial.print(", B:");
      mySerial.print(deathCount);
      mySerial.print(", O:");
      mySerial.print(RFIDcount);
      newLine();
      mySerial.print("R:");
      mySerial.print(TIEcount);
      mySerial.print(", G:");
      mySerial.print(blockCount);
      newLine();
      delay(5000000); // delay so no possibility of looping around
    }

    servoLeft.writeMicroseconds(1450); // keep moving past hash if not stopping
    servoRight.writeMicroseconds(1550);
    delay(1000);
```

```
        }

        if (qti2 < 500 && qti3 < 500) { // if mid QTIs see white
          servoLeft.writeMicroseconds(1460); // go forward
          servoRight.writeMicroseconds(1540);
        }

        if (qti2 > 500 && qti3 < 500) { // if left-mid QTI sees black, but not right-mid QTI
          servoLeft.writeMicroseconds(1480); // self-correct by going to the left a little
          servoRight.writeMicroseconds(1480);
        }

        if (qti2 < 500 && qti3 > 500) { // if right-mid QTI sees black, but not left-mid QTI
          servoLeft.writeMicroseconds(1520); // self-correct by going to the right a little
          servoRight.writeMicroseconds(1520);
        }
      }
    }
    else {
      delay(1000);
      qtiRFID = rcTime(2); // sets up the rfid on the side; logic is that if counter isn't 6, then
scan the rfid on the side
      qtiRFID = rcTime(2); // reads again for good measure
      if (qtiRFID > 500) {
        RFIDcount++; // increase RFID count
        digitalWrite(9, HIGH); // if the rfid is black, then light up the yellow led, if not then
don't turn it on
        delay(1000);
        digitalWrite(9, LOW); // turn the yellow led back off
      }
      servoLeft.writeMicroseconds(1450);         // keep going after reading the rfid
      servoRight.writeMicroseconds(1550);
      delay(500);
    }
  }
}

void xbeeReceive() {
  if (Xbee.available()) { // Is data available from XBee?
    char incoming = Xbee.read() ; // Read incoming character
    if (incoming == 'Y') { // if receive Y, add one to turret count
      turretCount++;
    }
    if (incoming == 'B') { // if receive B, add one to death star count
      deathCount++;
    }
    if (incoming == 'R') { // if receive R, add one to tie fighter count
      TIEcount++;
    }
    if (incoming == 'G') { // if receive G, add one to block count
```

```
      blockCount++;
    }
    if (incoming == 'D') { // if receive D, add one to bigD
      bigD++;
    }
    digitalWrite(8, HIGH); // turn on green receiving LED
    delay(100);
    digitalWrite(8, LOW); // turn off LED
  }
  delay (5) ; // reset
}

void xbeeSend(char outgoing) {
  Xbee.print(outgoing); // print to serial
  digitalWrite(3, HIGH); // turn on red outgoing LED
  delay(1000);
  digitalWrite(3, LOW); // turn off LED
  delay(500);
}

long rcTime(int pin)
{
  pinMode(pin, OUTPUT);                    // charge capacitor
  digitalWrite(pin, HIGH);                 // ..by setting pin ouput-high
  delay(1);                                // ..for 5 ms
  pinMode(pin, INPUT);                     // Set pin to input
  digitalWrite(pin, LOW);                  // ..with no pullup
  long time  = micros();                   // Mark the time
  while (digitalRead(pin));                // Wait for voltage < threshold
  time = micros() - time;                  // Calculate decay time
  return time;                             // Return decay time
}

void detachServos() {
  servoLeft.detach(); // detach left servo
  servoRight.detach(); // detach right servo
}

void attachServos() {
  servoLeft.attach(13); // reattach left servo
  servoRight.attach(12); // reattach right servo
}

void clearLCD() {
  mySerial.write(12); // clear LCD
}

void newLine() {
  mySerial.write(13); // create new line in LCD display
}
```

APPENDIX B: COST ESTIMATIONS

B.1 Initial Trade Study

1. Define Problem: Determine the Best Sensor to Locate the Position of the Death Star
2. Possible Sensors
    a. Compass Module 3 Axis HMC
    b. ColorPAL
    c. Photoresistor
    d. QTI Sensor
3. Trade Factors
    a. Cost
    b. Task Specific Sensing Accuracy
    c. Reliability
    d. Hassle

4. Specifications for Each Sensor

Sensor	**Cost ($)**	**Task Specific Sensing Accuracy**	**Reliability**	**Hassle**
https://www.parallax.com	4.99	Perfect	95%	Some
Photoresistor	2.99	Very Little	95%	A lot
QTI Sensor	9.99	Minimal	95%	A lot
ColorPAL	19.99	Excellent	95%	Some

Normalized Value	10	9	8	7	6	5	4	3	2	1
Cost ($)	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45	45-50
Task Specific Sensing Accuracy	Perfect	Excellent	Good	Above Average	Adequate	Sub Par	Some	Minimal	Very Little	None
Reliability	95-100%	90-95%	85-90%	80- 85%	75-80%	70-75%	65-70%	60-65%	55- 60%	50-55 %
Hassle	none	Almost none	Very little	Little	Some	Good Bit	A lot	Too much	Way too much	Unbereable

	Multiplication Factor	Rationale
Cost	1	Not really important - Duke's paying for it
Task Specific Sensing Accuracy	5	Very Important - Choosing the right sensor is the objective of the challenge
Reliability	3	Important - We need it to work for demonstrations
Hassle	3	Important - we would prefer to have sensors that are easy to set up and code

		Compass Module 3 Axis HMC		**Photoresistor**		**QTI Sensor**		**ColorPAL**	
	Weight Factor	Norm. Value	Total	Norm Value	Total	Norm. Value	Total	Norm Value	Total
Cost	1	10	10	10	10	10	10	7	7
Accuracy	5	10	50	2	10	3	15	9	45
Reliability	3	10	30	10	30	10	30	10	30
Hassle	3	6	18	4	12	4	12	6	18
GRAND TOTAL			108		62		67		100

Study Conclusion: Use the Compass Module 3 Axis HMC, potentially with the ColorPAL as a backup

B.2: Final Cost Estimation

Item	**Amount**	**Cost (dollars) /unit**
Resistor	1	0.1
Resistor	1	0.1
Resistor	1	0.1

Red LED	1	0.32
Yellow LED	1	0.32
Green LED	1	0.32
3/8" 4-40 pan head screw	4	0.02
1/2" round stainless steel spacer	4	1.3
BOE- Bot plastic wheel with tire(each)	2	$3.99
BOE-Bot 1" tail ball wheel (each)	1	3.95
BOE-Bot aluminum chassis (each)	1	24.99
BOE-Bot Li Ion Power Pack with cable and barrel plug (each)	1	$49.99
Li Ion cell Parallax, Inc.	1	$8.99
2A Fuse Digikey	1	$1.12
1 3/8" x 2" (5.1 x 3.5 cm) solderless breadboard	2	$3.49
Standard servomotor (each)	2	$12.99
Continuous rotation servomotor (each)	2	$13.99
Arduino ATMEGA 2560	1	$51.91
Board of Education Shield for Arduino	1	$39.99
USB A to B Cable Parallax	1	$4.99
Xbee Module	1	$29.99
7.5v 1A power supply	1	$14.99
Clear Tape	0.5 ft	$0.05
3-pin m/m header (each)	7	$0.50
Wires	0.1 pack	$7.95

Final Cost	**$310.69**

REFERENCES

*All circuit diagrams were taken from Parallax.com

“Parallax Robotics.” *Parallax Inc.* N.p., n.d. Web. 9 December 2017.